Findability through Traceability
A Realistic Application of Candidate Trace Links?

Markus Borg
Dept. of Computer Science, Lund University, Lund, Sweden

Keywords: Traceability, Impact Analysis, Information Seeking, Findability, Human Computer Interaction, Automation.

Abstract: Since software development is of a dynamic nature, the impact analysis is an inevitable work task. Traceability is known as one factor that supports this task, and several researchers have proposed traceability recovery tools to propose trace links in an existing system. However, these semi-automatic tools have not yet proven useful in industrial applications. Based on an established automation model, we analyzed the potential value of such a tool. We based our analysis on a pilot case study of an impact analysis process in a safety-critical development context, and argue that traceability recovery should be considered an investment in findability. Moreover, several risks involved in an increased level of impact analysis automation are already plaguing the state-of-practice work flow. Consequently, deploying a traceability recovery tool involves a lower degree of change than has previously been acknowledged.

1 INTRODUCTION

Change is an inherent characteristic of the evolution of large software systems. Consequently, *impact analysis*, the process of determining possible effects of proposed software changes, is an inevitable work task. Conducting an impact analysis is often a labor-intensive manual process, avoided unless absolutely necessary (Bohner, 2002). However, in development projects governed by safety regulations, impact analysis is a fundamental part of the development process, necessary for safety certification of products (IEC, 2003).

The impact analysis work task involves a high degree of information seeking, an increasingly costly activity among knowledge workers in general (Karr-Wisniewski and Lu, 2010). Previous studies have identified this issue also in software engineering projects (Olsson, 2002; Sabaliauskaite et al., 2010). As a large software development project constitutes a complex information landscape (i.e., thousands of artifacts such as requirements, source files, test cases and user manuals), analysing change impact is a challenging task. Thus, an important aspect of a development project is the *findability* it offers, defined as "the degree to which a system or environment supports navigation and retrieval" (Morville, 2005).

One way to support the information seeking is to maintain *traceability*, defined as "the degree to which artifacts are related" (IEEE Computer Society, 1990). It is widely recognized as an important factor for efficient software development (Antoniol et al., 2002; Domges and Pohl, 1998). However, maintaining trace links in an evolving system is a tedious task. To support this activity, several researchers have proposed *traceability recovery* (i.e., proposing trace links among existing artifacts) based on Information Retrieval (IR) approaches (Antoniol et al., 2002; Marcus and Maletic, 2003). However, despite numerous related publications during the last decade, success stories in industrial settings are conspicuously few (Borg et al., 2012). This makes us believe that the general expectations on the approach are too high, and that the tools should be considered from a new perspective.

The typical functionality of traceability recovery tools is to present the user a ranked list of candidate trace links, who then gets to vet the output (De Lucia et al., 2012). Since the human still is expected in the process, tools of this kind often claim to provide "semi-automatic" support. As automation is defined as "a device or system that accomplishes (partially or fully) a function that was previously, or conceivably could be, carried out (partially or fully) by a human" (Parasuraman et al., 2000), one might wonder to what level the tools actually automate human work. This leads us to discuss findability around the following research questions:

- RQ1: What type and level of automation do state-

Borg M..
Findability through Traceability - A Realistic Application of Candidate Trace Links?.
DOI: 10.5220/0004093501730181
In *Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE-2012), pages 173-181
ISBN: 978-989-8565-13-6

of-the-art (SoA) traceability recovery tools offer?

- RQ2: How would the introduction of a SoA traceability recovery tool change the state-of-practice (SoP) impact analysis work flow?

To tackle these questions, we conducted a pilot case study of an impact analysis process in a safety-critical development context, and assessed how SoA tool support could be applied. To structure our analysis, we applied the SWELL Automation Analysis Framework (SAAF), developed as an extension to an automation model initially proposed by Parasuraman *et al.* (Parasuraman et al., 2000). Based on our findings, we argue that proposed traceability recovery tools primarily should be considered as a step towards improved findability, rather than as an attempt to generate a full set of traces.

This paper is organized as follows: Section 2 presents work related to IR-based traceability recovery and automation analyses. Section 3 describes our case study and SAAF. Section 4 reports the outcome of our analysis, and finally Section 5 concludes and outlines future work.

2 RELATED WORK

2.1 IR-based Traceability Recovery

Several researchers have proposed expressing traceability recovery as an IR problem. Most developed traceability recovery tools implement standard IR techniques based on algebraic or probabilistic models (Antoniol et al., 2002; De Lucia et al., 2005; Marcus and Maletic, 2003). In such tools, the answer to a query is a ranked list of artifact suggestions, sorted by the level of calculated similarity (algebraic models), or probability that they are related (probabilistic models). The ranked list is analogous to the output of web search engines and enterprise search tools. Consequently, search results can be either relevant or non-relevant to the information need of the specific user.

A number of traceability recovery tools were developed as plug-ins. Klock *et al.* have developed *Traceclipse*, supporting trace link recovery and management within Eclipse (Klock et al., 2011). They developed Traceclipse to be expandable, to simplify meeting future feature requests and to easily support other IR models. The functionality of the plug-in was initially evaluated, however only tool output was considered rather than human-tool interaction. Canfora and Cerulo developed *Jimpa*, another traceability recovery plug-in for Eclipse (Canfora and Cerulo, 2006). They implemented probabilistic retrieval to establish links between change requests and source code, and evaluated the approach on three open source systems. De Lucia *et al.* developed their own Document Management Systen (DocMS), *ADAMS*, and the IR-based traceability recovery plug-in *ReTrace* (De Lucia et al., 2005). Furthermore, they have evaluated their plug-in in studies with student subjects (De Lucia et al., 2009). Falessi *et al.* implemented a plug-in, *PROUD*, to the industrial CASE tool *Enterprise Architect* (Falessi and Briand, 2009), and evaluated it in a controlled experiment with students and in an industrial case study. We, on the other hand, have proposed developing a traceability recovery plug-in to HP Quality Center (Borg, 2011a; Borg, 2011b). Developing plug-ins to tools already deployed in industry enables in-vivo studies without introducing additional external tools.

2.2 Automation Analysis

Several taxonomies and frameworks have been developed to support the analysis of automation. A comprehensive overview, however from the viewpoint of manufacturing, was recently presented by Frohm *et al.* (Frohm et al., 2008). In their work, they report eight different definitions of "levels of automation". Sheridan and Verplank developed a 10-level taxonomy for automation levels in 1980 (Sheridan and Verplank, 1978). Billings studied automation in the context of air-traffic controllers (Billings, 1997). His work explicitly separated automation and human functions, and defined a continuum of management modes from unassisted control to fully autonomous operations. Parasuraman *et al.* developed a model incorporating both the type and the level of automation (Parasuraman et al., 2000). We extend this model by adding two preceding and one subsequent analysis phases, described in detail in Section 4.3. The model by Parasuraman *et al.* was previously applied in human-computer interaction research to analyze adaptive automation solutions for air-traffic control (Clamann et al., 2002). However, to the best of our knowledge, it has not been applied to analyze software engineering tools.

Huffman Hayes *et al.* touched upon the automation questions for traceability recovery in a technical report (Huffman Hayes et al., 2006), in which they base the discussions on their long experience of hands-on traceability activities in industry. Furthermore, they have published several studies on how engineers should work with semi-automatic tool support, including how humans interact with tools in the traceability loop (Huffman Hayes and Dekhtyar,

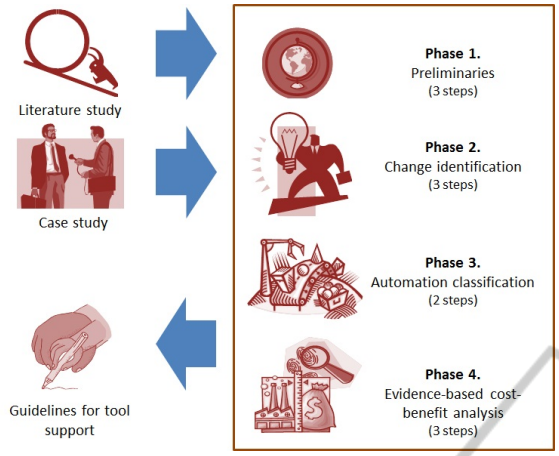

Figure 1: Overview of the automation analysis. The box represents SAAF.

2005). However, the automation analysis is not the central part of their publications, which motivated our inquiry.

3 METHOD

To concretize our discussion on automation, we applied our automation analysis on a specific case in a safety-critical development context. An overview of SAAF, the framework used for the automation analysis, is presented in Figure 1. It is based on the model by Parasuraman *et al.* (Parasuraman et al., 2000) mentioned in Section 2.2. Our understanding of tool support for traceability recovery originates from an extensive literature review, and the outcome of this automation analysis is intended to guide our future tool developing efforts.

3.1 Safety-critical Impact Analysis - A Pilot Study

To better understand how traceability recovery can support the impact analysis process, we developed an initial model of the inherent information seeking activity based on our industrial experiences. To validate the model, we presented it to three software engineers from the case company. We communicated primarily via e-mail, and the respondents were selected using convenience sampling. However, to improve generalizability, we selected respondents representing three different development teams from two different departments. Based on the feedback received, we refined the model to the version presented in Section 4.2. Furthermore, we asked the respondents about their views on risks involved in increasing automation in the impact analysis process.

3.2 Phases of SAAF

The first phase, *Preliminaries*, establishes the focus, i.e., the scope of activities affected as well as effect targets, for the automation effort within a context. Also, the phase clarifies any assumptions taken and what is included in the analysis. The three steps of this phase describe: *Context of the automation*, *Scope of the automation*, and *Effect targets*.

Automation change identification, the second phase of SAAF, describes pre- and post-automation task flows. This phase should specify which work tasks are changed, added or removed as a consequence of automation. The three steps of this phase describe: *Pre-automation work flow*, *Post-automation work flow*, and *Changed/Added/Removed tasks*.

Automation classification, third phase, analyses automation according to the model by Parasuraman *et al.* (Parasuraman et al., 2000). The object of the analysis is both the pre- and post-automation work flows. The two steps of this phase comprise analysis of: *Types of automation*, and *Levels of automation* (presented in Table 1 and Figure 3).

The final phase, *Automation impact analysis*, estimates both direct and indirect effects of the increased automation. Since automated solutions typically bring both positive and negative effects, understanding them prior to implementing any changes is essential. Our analysis targets threats involved in the actual automation, and also its cognitive side-effects. The final phase of SAAF also includes a break-even analysis, a parametric assessment of benefits, where the parameter values are selected to equate costs and benefits. It is one of the methods that evolved to quantify benefits of information systems (Sassone, 1988). Fixed costs and costs dependant on volume are compared to determine the volume at which an automation investment results in neither a profit nor a loss, the so called *breakpoint*. We base discussions on the feasibility of traceability recovery for impact analysis on such an initial analysis, and report: *Direct effects*, *Indirect effects*, and *Existing evidence for effects*.

4 RESULTS AND DISCUSSION

This section describes the results from applying SAAF. Every phase is concluded by our final assessment, expressed using the levels presented in Table 1.

Table 1: Summary of types and levels of automation (Parasuraman et al., 2000).

Type	Level	Description
Information	10	Autonomous, ignoring human.
acquisition	9	May inform human.
⇓	8	Informs human if asked.
Information	7	Executes automatically, then informs.
analysis	6	Allows human time to veto.
⇓	5	Executes if human approves.
Decision	4	Suggests one alternative.
selection	3	Narrows down selection to a few.
⇓	2	Offers complete set of alternatives.
Action	1	No assistance at all.
implementation		

4.1 Preliminaries

4.1.1 Description of the Context

The analyzed impact analysis process originates from a large multinational company active in the power and automation sector. The development context is safety-critical embedded development in the domain of industrial control systems, governed by IEC 61511 (IEC, 2003). The number of developers is in the magnitude of hundreds; a project has typically a length of 12-18 months and follows an iterative stage-gate project management model. The software is certified to a Safety Integrity Level (SIL) of 2 as defined by IEC 61508 (IEC, 2010), corresponding to a risk reduction factor of 1,000,000-10,000,000 for continuous operation. There are process requirements on the maintenance of traceability information, especially between requirements and test cases. Both requirements and test case descriptions are predominantly specified in English natural language text.

4.1.2 Description of the Work Task

As specified in IEC 61511 (IEC, 2003), impact of proposed software changes should be analyzed before implementation. In the studied case, this process is tightly integrated in the *Defect Management System* (DefMS). The issues in the DefMS, i.e., defect reports and change requests, are administered by a change control board. The board distributes issues to responsible teams for investigation. As part of the investigation, developers are required to perform an impact analysis, and report their results according to a project specific template. The template, developed by internal safety engineers and validated by an external certifying agency, contains between 5-20 questions depending on the SIL of the affected software components. In the template, several questions explicitly ask for trace links. The developer is required to specify source code that will be modified (with a class-level granularity), and also which related software artifacts need to be updated to reflect the changes, e.g., requirement specifications, design documentation, test case descriptions, test scripts and user manuals. Furthermore, the report should specify which high-level system requirements cover the involved features, and which test cases should be executed to verify that the changes are correct once implemented in the system. The test case selection should cover both developer-centric functional testing, and system testing conducted by the test organization.

4.1.3 Effect Targets

The intention of increased automation is to make the impact analysis faster and more accurate. Also, one engineer explained “It is important to reduce the number of mundane questions to save the effort for the ones requiring thought and analytical abilities”.

4.2 Automation Change Identification (RQ2)

4.2.1 Pre-automation Task Flow

A major part of the impact analysis involves specifying trace links to related software artifacts. As there rarely are any requirement traceability matrices to consult, the tracing is mainly a poorly supported information seeking activity. If the engineers do not already know which artifacts (if any) are related to the issue, or if they do not know where to find the information, they have to search or browse databases or seek information from colleagues. When this information need arise, a typical first step is to search the DefMS for already solved issues that are similar, as presented in Figure 2. If no such issues are found, one could search the DocMS for relevant project documentation, or ask a colleague for help as a last resort. One engineer stated “I probably search for information in project documentation more than I should, it is

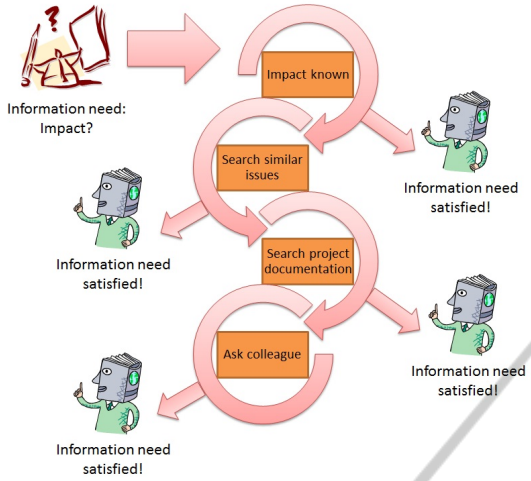

Figure 2: Overview of the SoP impact analysis process.

very time-consuming and rarely successful".

4.2.2 Post-automation Task Flow

The idea of the traceability recovery tool is to support the two steps of manual database searching by automatically executing search queries. Without human action, we envision the tool to search for both related issues in the DefMS and related documentation in the DocMS. Based on the textual content of the currently analyzed issue, the tool predicts which software artifacts are the most likely to be related. As in the manual work flow, the last resort is to ask a colleague.

4.2.3 Changed/Added/Removed Tasks

From the perspective of the engineer, the work task is slightly altered. An automatic search is conducted and the resulting search hits, i.e., candidate trace links, are presented. Thus, an added human subtask is to assess the search hits. If the search result is enough to satisfy the information need, the manual DefMS and DocMS searching are removed subtasks. On the other hand, if the engineer does not consider the search results to provide enough information, the engineer will as before have to seek information by manual queries, or by asking a colleague.

4.3 Automation Classification (RQ1)

4.3.1 Types of Automation

Table 1 shows Parasuraman *et al.*'s types of automation. The first step, *Information acquisition*, refers to operations supporting human sensory processes, sensing and registration of input data. In the case of traceability recovery, the required information is stored digitally in databases. Software artifacts are typically distributed in separate systems with poor interoperability, a condition that applies also to the case we study. Since the usefulness of an IR-based traceability recovery tool is dependant on which software artifacts it can access and index, plug-in solutions to existing DefMS and DocMS have the advantage of being deployed where the information actually resides.

Information analysis, the second step of an automation solution, deals with the cognitive functions such as working memory and inferential processes (Parasuraman et al., 2000). Regarding IR-based traceability recovery tools, it is not meaningful to distinguish between acquisition and analysis of information. As the information is accessed, it is also analyzed. This includes the steps of the implemented IR model such as preprocessing (stop word removal, stemming etc.), feature extraction and weighting, and extraction of language models. In the rest of this report, we consider information acquisition and analysis to be one single type of automation.

The third step, *Decision selection*, comprises the augmentation and replacement of human decision options. Supporting decision selection is the backbone of IR-based traceability recovery tools, since they are intended to present candidate trace links. The tools rank search results, and present them to the user. Furthermore, a number of traceability recovery tools offer the user both filtering and highlighting.

Action implementation, the last automation type, is defined as the actual machine execution of the action choice. For the IR-based traceability recovery plug-in we envision, the action implementation is limited to correctly reporting the outcome in the impact analysis template described in Section 4.1.2. Automating this step is meaningful since it would reduce the risk of human input errors (which are known to exist), e.g., incorrect use of document identifiers, and copy/paste errors.

4.3.2 Levels of Automation

Table 1 shows the levels of automation according to Parasuraman *et al.*, from no support at all to a fully autonomous solution. Based on our analysis of types of automation, we studied the corresponding levels of automation, and the risks involved in increasing them.

Regarding *Information acquisition and analysis*, the SoP activity is to manually input search queries in search tools offered by DefMSs and DocMSs. In such tools, the human activity is at best supported by a set of boolean search operations, e.g., AND, OR, NOT. The SoA traceability recovery tools on the other hand, automatically enters and executes search queries. The human is not at all involved in this process, however

error messages tend to appear if major failures occur. Letting the tools access too much information is a security issue. In many cases, employees have different access rights. Efforts to improve information access in enterprise search solutions are often limited by policy decisions (Tolone et al., 2005).

The SoP activities corresponding to the automation type *Decision support* are mainly concerned with vetting various search results. Search strings can be refined until the information need of the engineer is satisfied. SoP search solutions typically return a ranked list of documents and support features such as sorting and filtering. More advanced search solutions also implement features as *query expansion* and *relevance feedback* (Baeza-Yates and Ribeiro-Neto, 2011).

After automatic execution of the search queries, SoA tracebility recovery tools do not differ from SoP search solutions. In both cases, the goal is to limit the search space of the engineers, and to at least give them starting points for browsing to the information they are seeking. Obviously, an ideal traceability recovery tool would automatically make decisions with a better judgement than a human engineer. Instead, since search results include both relevant and non-relevant results, increasing the automation to levels where the human is not part of the process leads to both false positives and missed trace links. Since the accuracy of SoA traceability recovery is considered low, fully removing the human involvement is currently not feasible (Oliveto et al., 2007; De Lucia et al., 2012). IR tools balance on the precision-recall trade-off (Baeza-Yates and Ribeiro-Neto, 2011), and a fully automated solution have to be designed with care. Missed links threat underestimating change impact, which motivates search tools offering a high recall. On the other hand, false positives caused by low precision force engineers to spend extra effort. For both cases, incorrect effort estimations are consequences.

The last automation type in the sequence, the *Action implementation*, is currently supported by a template and corresponding guidelines developed by internal safety engineers. The outcome of the impact analysis is reported manually, by entering free text in a document. The tailored traceability recovery plug-in we envision, enables simple drag-and-drop operations in a graphical user interface. Both manually typing free-text information in a template and operations in a graphical interface introduce errors due to the human factor, thus we do not consider this a major risk. Although, efficient UIs enable engineers to input incorrect information faster. Consequently, an increased level of automation of the last automation type in the sequence might actually increase the number of human errors.

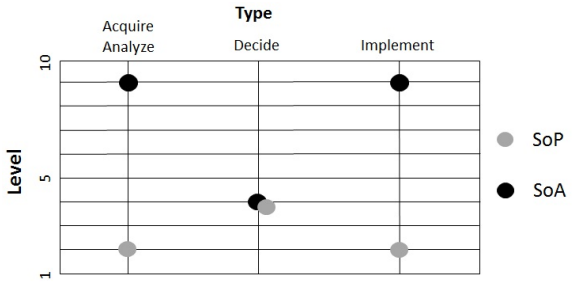

Figure 3: Types and levels of automation for IR-based traceability recovery support in impact analysis. The major risks involved in increasing the levels are from left to right: contravening access policies, incorrect impact analysis, increased number of human errors.

4.4 Automation Impact Analysis (RQ2)

4.4.1 Direct Effects

There are two main hypotheses of increasing the level of automation by deploying a traceability recovery plug-in. First, engineers will on average finish the impact analysis work task faster. Second, the correctness of the engineers' impact analyses will be higher.

4.4.2 Indirect Effects

Besides risks already mentioned, there is a risk that additional tool support, if it is well received, would make engineers too confident in their output. Such overconfidence might cause engineers, especially under stress, to hastily accept tool output as final answers. Another risk of deploying efficient search support is that communication between developers might decrease, as people might rely more on tools.

4.4.3 Evidence of Effects

There are few evaluations of deploying IR-based traceability recovery tools in complete software development projects, and only one of them was conducted in an industrial setting. Li *et al.* conducted a case study on impact analysis in a five-people project running in a Chinese company for 30 weeks (Li et al., 2008). They concluded it to be a feasible approach, and that it helped engineers finish the tracing tasks faster. De Lucia *et al.* conducted another case study, however in a university setting (De Lucia et al., 2009). By studying seven student development projects, they found that deploying their IR-based traceability recovery plug-in improved the maintenance of trace-

ability information, as more trace links were discovered.

Apart from case studies, several controlled experiments have concluded that tools implementing IR-based traceability recovery can be beneficial. For similar tasks related to establishing trace links, both Huffman Hayes *et al.* (Huffman Hayes et al., 2007) and De Lucia *et al.* (De Lucia et al., 2009) concluded that working with tool support is better than working manually, and that it improves accuracy and/or efficiency. De Lucia *et al.* also found that inexperienced developers benefit most from such tool support (De Lucia et al., 2009). On the contrary, an experiment by Falessi *et al.* concluded that letting student subjects work with their IR-traceability recovery tool did not lead to any significant advantages (Falessi and Briand, 2009). Also, in contrast to findings by De Lucia *et al.* regarding impact of experience, they found the tool support more useful when used by a requirements analyst in an Italian company.

The usefulness of IR solutions are known to depends on the context, users, and the task it is meant to support, which is also indicated by the conclusions from previous studies. Consequently, the applicability of deploying a tailored plug-in in the safety-critical context of our case study is uncertain. However, primarily considering it as an investment in findability (i.e., moving beyond manual keyword searching to automatically suggesting search results), we expect the information seeking to be more efficient. Figure 4 presents a break-even analysis, assuming that the tool support would make an engineer complete an impact analysis faster. The main costs in an automated solution would be initial, as the tool maintenance is expected to be negligible. Instead the costs are mainly related to development and deployment of a tool, and user training. The tracing costs would then linearly increase depending on the number of tracing tasks performed. For the manual tracing process, the tracing cost is expected to decrease as the number of tasks grow due to human learning effects. Thus, for a traceability recovery plug-in to be a meaningful investment, we would expect two aspects from the development context. First, the information landscape must be challenging enough to make tracing a time-consuming task (high slope of manual curve). Second, the tracing task must be common enough to be considered an issue (*n* high enough for curves to intersect). Whether the studied case fulfils these aspects requires further study.

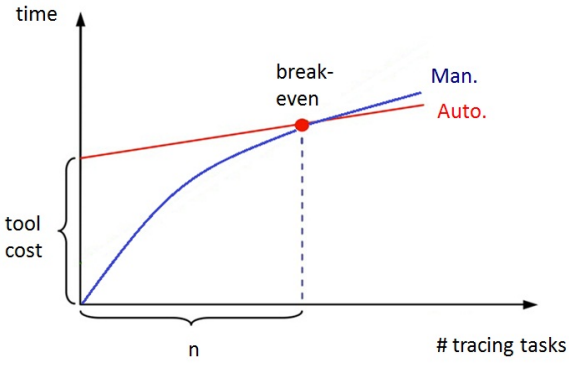

Figure 4: Initial break-even analysis for automated impact analysis.

5 CONCLUSIONS AND FUTURE WORK

We have presented a structured analysis of how a SoA IR-based traceability recovery tool could support findability in an impact analysis work task. Also, we argue that the tools should be judged as search tools rather than as traceability miners, and thus should be evaluated accordingly. The rest of our contribution is threefold. First, we presented a model of the information seeking activity involved in an impact analysis work task in a safety-critical development context. Second, we found that deploying a SoA traceability recovery tool, with a limited risk, can increase the level of automation, but not for the automation type defined as decision selection. Third, based on an initial break-even analysis, we claim that investing in automated traceability recovery only is worth the effort if both (a) the information landscape is challenging, and (b) the artifact tracing is a frequent work task.

Future work includes implementing an IR-based traceability recovery plug-in to HP Quality Center and evaluating it in an industrial context, based on established methodology for user evaluations in information retrieval. Furthermore, a deeper case study on impact analysis processes in safety-critical software engineering should be conducted. Such a study should be dominated by qualitative analysis, since quantitative analyses have dominated the field of IR-based traceability recovery.

ACKNOWLEDGEMENTS

This study was done as part of the SWELL[1] course on Automated Verification and Validation where we have

[1] swell.se

studied and developed models for analyzing levels of automation in V&V. Thanks go to Rickard Torkar, Saïd Assar, and Per Runeson for review comments.

REFERENCES

Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., and Merlo, E. (2002). Recovering traceability links between code and documentation. In *Trans. on Software Engineering*, volume 28, pages 970–983.

Baeza-Yates, R. and Ribeiro-Neto, B. (2011). *Modern Information Retrieval: The Concepts and Technology behind Search*. Addison-Wesley.

Billings, C. (1997). *Aviation Automation: The Search for a Human-Centered Approach*. Larrence Erlbaum Associates, New Jersey.

Bohner, S. (2002). Software change impacts-an evolving perspective. In *Proceedings of the International Conference on Software Maintenance*, pages 263– 272.

Borg, M. (2011a). In vivo evaluation of large-scale IR-based traceability recovery. In *Proceedings of the European Conference on Software Maintenance and Reengineering*, pages 365–368.

Borg, M. (2011b). IR-based traceability recovery as a plugin - an industrial case study. In *Fourth BCS-IRSG Symposium on Future Directions in Information Access*.

Borg, M., Wnuk, K., and Pfahl, D. (2012). Industrial comparability of student artifacts in traceability recovery research - an exploratory survey. In *Proceedings of the 16th European Conference on Software Maintenance and Reengineering*.

Canfora, G. and Cerulo, L. (2006). Fine grained indexing of software repositories to support impact analysis. In *Proceedings of the International Workshop on Mining software repositories*, pages 105–111.

Clamann, M., Wright, M., and Kaber, D. (2002). Comparison of performance effects of adaptive automation applied to various stages of human-machine system information processing. In *Proc. of the Ann. Meeting of the Human Factors and Ergonomics Soc.*, pages 342–346.

De Lucia, A., Fasano, F., Oliveto, R., and Tortora, G. (2005). ADAMS re-trace: A traceability recovery tool. In *Proc. of the 9th European Conference on Software Maintenance and Reengineering*, pages 32–41.

De Lucia, A., Marcus, A., Oliveto, R., and Poshyvanyk, D. (2012). Information retrieval methods for automated traceability recovery. In Cleland-Huang, J., Gotel, O., and Zisman, A., editors, *Software and Systems Traceability*, pages 71–98. Springer, London.

De Lucia, A., Oliveto, R., and Tortora, G. (2009). Assessing IR-based traceability recovery tools through controlled experiments. *Empirical Software Engineering*, 14(1):57–92.

Domges, R. and Pohl, K. (1998). Adapting traceability environments to project-specific needs. *Communications of the ACM*, 41(12):54–62.

Falessi, D. and Briand, L. (2009). The impact of automated support for linking equivalent requirements based on similarity measures. Technical report, Simula.

Frohm, J., Lindstrom, V., Winroth, M., and Stahre, M. (2008). Levels of automation in manufacturing. *Ergonomia*, page 29.

Huffman Hayes, J. and Dekhtyar, A. (2005). Humans in the traceability loop: can't live with 'em, can't live without 'em. In *Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering*, pages 20–23.

Huffman Hayes, J., Dekhtyar, A., and Sundaram, S. (2006). Advances in dynamic generation of traceability links: Two steps closer to full automation? Technical report, University of Kentucky.

Huffman Hayes, J., Dekhtyar, A., Sundaram, S., Holbrook, A., Vadlamudi, S., and April, A. (2007). REquirements TRacing on target (RETRO): improving software maintenance through traceability recovery. *Innovations in Systems and Software Engineering*, 3(3):193–202.

IEC (2003). IEC 61511-1 ed 1.0, safety instrumented systems for the process industry sector.

IEC (2010). IEC 61508 ed 2.0, Electrical/Electronic/Programmable electronic safety-related systems.

IEEE Computer Society (1990). 610.12-1990 IEEE standard glossary of software engineering terminology. Technical report.

Karr-Wisniewski, P. and Lu, Y. (2010). When more is too much: Operationalizing technology overload and exploring its impact on knowledge worker productivity. *Computers in Human Behavior*, 26(5):1061–1072.

Klock, S., Gethers, M., Dit, B., and Poshyvanyk, D. (2011). Traceclipse: an eclipse plug-in for traceability link recovery and management. In *Proceeding of the 6th International Workshop on Traceability in Emerging Forms of Software Eengineering*, pages 24–30.

Li, Y., Li, J., Yang, Y., and Li, M. (2008). Requirement-centric traceability for change impact analysis: A case study. In *International Conference on Software Process*, pages 100–111.

Marcus, A. and Maletic, J. (2003). Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proc. of the Int'l Conference on Software Engineering*, pages 125–135.

Morville, P. (2005). *Ambient Findability: What We Find Changes Who We Become*. O'Reilly Media.

Oliveto, R., Antoniol, G., Marcus, A., and Hayes, J. (2007). Software artefact traceability: the Never-Ending challenge. In *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, pages 485–488.

Olsson, T. (2002). *Software Information Management in Requirements and Test Documentation*. Licentiate thesis, Lund University.

Parasuraman, R., Sheridan, T., and Wickens, C. (2000). A model for types and levels of human interaction with automation. *Transactions on Systems, Man and Cybernetics*, 30(3):286–297.

Sabaliauskaite, G., Loconsole, A., Engstrm, E., Unterkalmsteiner, M., Regnell, B., Runeson, P., Gorschek, T., and Feldt, R. (2010). Challenges in aligning requirements engineering and verification in a Large-Scale industrial context. In *Requirements Engineering: Foundation for Software Quality*, pages 128–142.

Sassone, P. (1988). Cost benefit analysis of information systems: a survey of methodologies. In *Proceedings of the Conference on Office Information Systems*, pages 126–133.

Sheridan, T. and Verplank, W. (1978). Human and computer control of undersea teleoperators. Technical, MIT Man-Machine Systems Laboratory, Cambridge, MA, United states.

Tolone, W., Ahn, G., Pai, T., and Hong, S. (2005). Access control in collaborative systems. *ACM Computing Surveys*, 37(1):29–41.