Evaluation of Traceability Recovery in Context: A Taxonomy for Information Retrieval Tools

Borg, Markus; Runeson, Per; Brodén, Lina

Published in:
16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)

DOI:
10.1049/ic.2012.0014

2012

Evaluation of Traceability Recovery in Context: A Taxonomy for Information Retrieval Tools

Markus Borg, Per Runeson, Lina Brodén
Department of Computer Science
Lund University
Sweden
{markus.borg, per.runeson}@cs.lth.se

Abstract*—*Background: **Development of complex, software intensive systems generates large amounts of information. Several researchers have developed tools implementing information retrieval (IR) approaches to suggest traceability links among artifacts.** ***Aim:*** **We explore the consequences of the fact that a majority of the evaluations of such tools have been focused on benchmarking of mere tool output.** ***Method:*** **To illustrate this issue, we have adapted a framework of general IR evaluations to a context taxonomy specifically for IR-based traceability recovery. Furthermore, we evaluate a previously proposed experimental framework by conducting a study using two publicly available tools on two datasets originating from development of embedded software systems.** ***Results:*** **Our study shows that even though both datasets contain software artifacts from embedded development, the characteristics of the two datasets differ considerably, and consequently the traceability outcomes.** ***Conclusions:*** **To enable replications and secondary studies, we suggest that datasets should be thoroughly characterized in future studies on traceability recovery, especially when they can not be disclosed. Also, while we conclude that the experimental framework provides useful support, we argue that our proposed context taxonomy is a useful complement. Finally, we discuss how empirical evidence of the feasibility of IR-based traceability recovery can be strengthened in future research.**

I. Introduction

Large-scale software development generates large amounts of information. Enabling software engineers to efficiently navigate the document space of the development project is crucial. One typical way to structure the information within the software engineering industry is to maintain traceability, defined as "the ability to describe and follow the life of a requirement, in both a forward and backward direction" [18]. This is widely recognized as an important factor for efficient software engineering as it supports activities such as verification, change impact analysis, program comprehension, and software reuse [2].

Several researchers have proposed using information retrieval (IR) techniques to support maintenance of traceability information [28], [29], [23], [10]. Traceability can be maintained between any software artifacts, i.e., any piece of information produced and maintained during software development. Textual content in natural language (NL) is the common form of information representation in software engineering [31]. Tools implementing IR-based traceability recovery suggest traceability links based on textual similarities, for example between requirements and test cases. However, about 50% of the evaluations have been conducted using one of the four datasets available at the *Center of Excellence for Software Traceability* (COEST)[1][5]. Consequently, especially the *CM-1* and *EasyClinic* datasets have turned into de-facto benchmarks of IR-based traceability recovery, further amplified by "traceability challenges" issued by the *Traceability in Emerging Forms of Software Engineering* (TEFSE) workshop series.

Developing a repository of benchmarks for traceability research is a central part of COEST's vision, and thus has been discussed in several publications [12], [13], [7], [6]. As another contribution supporting benchmarking efforts and evaluations in general, Huffman Hayes *et al.* proposed an experimental framework for requirements tracing [21]. It is known that benchmarks can advance tool and technology development, which for instance has been the case for the Text REtrieval Conference (TREC), driving large-scale evaluations of IR methodologies [42]. On the other hand, benchmarking introduces a risk of over-engineering IR techniques on specific datasets. TREC enables generalizability by providing very large amounts of texts, but is still limited to certain domains such as news articles. Whether it is possible for the research community on traceability in software engineering to collect and distribute a similarly large dataset is an open question. Runeson *et al.* have discussed the meaning of benchmarks, however in relation to software testing [39]. They stress that researchers should first consider what the goal of the benchmarking is, and that a benchmark is a selected case that should be analyzed in its specific context. Consequently, as a benchmark is not an "average situation", they advocate that benchmarks should be studied with a qualitative focus rather than with the intention to reach statistical generalizability.

To enable categorization of evaluations of IR-based traceability recovery, including benchmarking studies, we adapted Ingwersen and Järvelins framework of IR evaluation contexts [24] into a taxonomy of IR-based traceability recovery evaluations. Within this context taxonomy, described in detail in Section II, typical benchmarking studies belong to the innermost context. In the opposite end of the context taxonomy, the outermost evaluation context, industrial case studies reside. Furthermore, we add a dimension of study environment to the context taxonomy, to better reflect the external validity of the

[1]COEST, http://www.coest.org/

studied project, and its software artifacts.

To illustrate the context taxonomy with an example, and to evaluate the experimental framework proposed by Huffman Hayes *et al.* [21], we have conducted a quasi-experiment using the tools RETRO [23] and ReqSimile [35], using two sets of proprietary software artifacts as input. This constitutes a replication of earlier evaluations of RETRO [44], [14], as well as an extension considering both tools and input data. Also, we describe how it would be possible to extend this study beyond the innermost level of the context taxonomy, i.e., moving beyond technology-oriented evaluation contexts. Finally, we discuss concrete ways to strengthen the empirical evidence of IR-based traceability tools in relation to findings and suggestions by Falessi *et al.* [16], Runeson *et al.* [39], and Borg *et al.* [4], [5].

The paper is organized as follows. Section II gives a short overview of related work on IR-based traceability recovery in software engineering, while Section III describes our context taxonomy. Section IV presents the research methodology, and the frameworks our experimentation follows. Section V presents the results from the experiments. In Section VI we discuss the implications of our results, the validity threats of our study, and evaluations of IR-based traceability recovery in general. Finally, Section VII presents conclusions and suggests directions of future research.

II. Background and Related work

During the last decade, several researchers proposed expressing traceability recovery as an IR problem. Proposed tools aim to support software engineers by presenting candidate traceability links ranked by textual similarities. The query is typically the textual content of a software artifact you want to link to other artifacts. Items in the search result can be either relevant or irrelevant, i.e., correct or incorrect traceability links are suggested.

In 2000, Antoniol *et al.* did pioneering work on traceability recovery when they used the standard *Vector Space Model* (VSM) [40] and the *binary independence model* [37] to suggest links between source code and documentation in natural language [1]. Marcus and Maletic introduced *Latent Semantic Indexing* (LSI) [11] to recover traceability in 2003 [31]. Common to those papers is that they have a technical focus and do not go beyond reporting precision-recall graphs.

Other studies reported evaluations of traceability recovery tools using humans. Huffman Hayes *et al.* developed a traceability recovery tool named RETRO and evaluated it using 30 student subjects [23]. The students were divided into two groups, one working with RETRO and the other working manually. Students working with the tool finished a requirements tracing task faster and with a higher recall than the manual group, the precision however was lower. Natt och Dag *et al.* developed the IR-based tool ReqSimile to support market-driven requirements engineering and evaluated it in a controlled experiment with 23 student subjects [35]. They reported that subjects supported by ReqSimile completed traceability related tasks faster than subjects working without any tool support. De Lucia *et al.* conducted a controlled experiment with 32 students on the usefulness of tool-supported traceability recovery [9] and also observed 150 students in 17 software development projects [10]. In line with previous findings, they found that subjects using their tool completed a task, related to tracing various software artifacts, faster and more accurately than subjects working manually. They concluded that letting students use IR-based tool support is helpful when maintenance of traceability information is a process requirement.

Several previous publications have contributed to advancing the research on IR-based traceability recovery, either by providing methodological advice, or by mapping previous research. Huffman Hayes and Dekhtyar published a framework intended to advance reporting and conducting of empirical experiments on traceability recovery [21]. However, the framework has unfortunately not been applied frequently in previous evaluations, and the quality of reporting varies [5]. Another publication that offers structure to IR-based traceability recovery, also by Huffman Hayes *et al.*, distinguishes between *studies of methods* (are the tools capable of providing accurate results fast?) and *studies of human analysts* (how do humans use the tool output?) [22]. These categories are in line with experimental guidelines by Wohlin *et al.* [48], where the types of experiments in software engineering are referred to as either technology-oriented or human-oriented. Moreover, Huffman Hayes *et al.* propose assessing the accuracy of tool output, wrt. precision and recall, according to quality intervals named *Acceptable*, *Good*, and *Excellent*, based on the first author's practical experience of requirements tracing. Also discussing evaluation methodology, a recent publication by Falessi *et al.* proposes seven empirical principles for evaluating the performance of IR techniques [16]. Their work covers study design, statistical guidelines, and interpretation of results. Also, they present implementation strategies for the seven principles, and exemplify them in a study on industrial software artifacts originating from an Italian company.

Going beyond the simplistic measures of precision and recall is necessary to evolve IR tools [27], thus measures such as *Mean Average Precision* (MAP) [30], and *Discounted Cumulative Gain* (DCG) [26] have been proposed. To address this matter in the specific domain of IR-based traceability, a number of so called *secondary measures* have been proposed. Sundaram *et al.* developed *DiffAR*, *DiffMR*, *Lag*, and *Selectivity* [45] to assess the quality of generated candidate links.

In the general field of IR research, Ingwersen and Järvelin argue that IR is always evaluated in a context [24]. Their work extends the standard methodology of IR evaluation, the Laboratory Model of IR Evaluation developed in the Cranfield tests in the 60s [8], challenged for its unrealistic lack of user involvement [27]. Ingwersen and Järvelin proposed a framework, *The Integrated Cognitive Research Framework*, consisting of four integrated evaluation contexts, as presented in Figure 1. The innermost *IR context*, referred to by Ingwersen and Järvelin as "the cave of IR evaluation", is the most frequently studied level, but also constitutes the most simplified

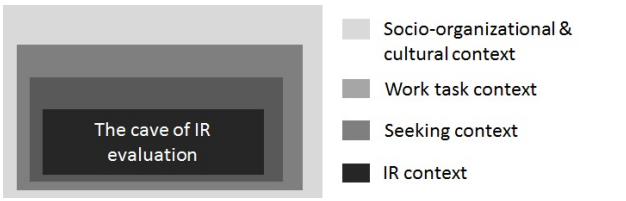

Fig. 1. The Integrated Cognitive Research Framework by Ingwersen and Järvelin [24], a framework for IR evaluations in context.

context. The *seeking context*, "drifting outside the cave", is used to study how users find relevant information among the information that is actually retrieved. The third context, the *work task context* introduces evaluations where the information seeking is part of a bigger work task. Finally, in the outermost realm, the *socio-organizational & cultural context*, Ingwersen and Järvelin argue that socio-cognitive factors are introduced, that should be studied in natural field experiments or studies, i.e., in-vivo evaluations are required. Moreover, they propose measures for the different evaluation contexts [24]. However, as those measures and the evaluation framework in itself are general and not tailored for neither software engineering nor traceability recovery, we present an adaptation in Section III.

III. Derivation of Context Taxonomy

Based on Ingwersen and Järvelin's framework [24], we introduce a four-level context taxonomy in which evaluations of IR-based traceability recovery can be conducted, see Table I. Also, we extend it by a dimension of evaluation environments motivated by our previous study [5], i.e., *proprietary*, *open source*, or *university*, as depicted in Figure 2. The figure also shows how the empirical evaluations presented in Section II map to the taxonomy. Note that several previous empirical evaluations of IR-based traceability recovery have used software artifacts from the open source domain, however, none of them were mentioned in Section II.

We refer to the four integrated contexts as the *retrieval context* (Level 1), the *seeking context* (Level 2), the *work task context* (Level 3), and the *project context* (Level 4). Typical benchmarking experiments [1], [31], [16], similar to what is conducted within TREC, reside in the innermost retrieval context. Accuracy of tool output is measured by the standard IR-measures precision, recall, and F-score (defined in Section IV-B2. In order to enable benchmarks in the seeking context, user studies or secondary measures are required [22], [45]. In both the two innermost contexts, traceability recovery evaluations are dominated by quantitative analysis. On the other hand, to study the findability offered by IR tools in the seeking context, defined as "the degree to which a system or environment supports navigation and retrieval" [34], researchers must introduce human subjects in the evaluations.

Regarding evaluations in the work task context, human subjects are necessary. Typically, IR-based traceability recovery in this context has been evaluated using controlled experiments with student subjects [35], [23], [9]. To assess the usefulness of tool support in work tasks involving traceability

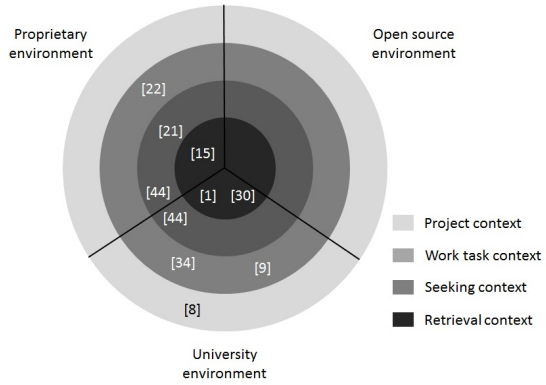

Fig. 2. Contexts and environments in evaluations of IR-based traceability recovery. The numbers refer to references. Note that the publication by Sundaram *et al.* [45] contains both an evaluation in an industrial environment, as well as an evaluation in the university environment.

recovery, realistic tasks such as requirements tracing, change impact analysis, and test case selection should be studied in a controlled, in-vitro environment. Finally, in the outermost project context, the effect of deploying IR-based traceability recovery tools should be studied in-vivo in software development projects. Due to the typically low level of control in such study environments, a suitable evaluation methodology is a case study. An alternative to industrial in-vivo evaluations is to study student development projects, as De Lucia *et al.* [10] have done. In the work task context both quantitative and qualitative studies are possible, but in the project context qualitative analysis dominates. As applied researchers we value technology-oriented evaluations in the retrieval and seeking contexts, however, our end goal is to study IR tools in the full complexity of an industrial environment.

IV. Method

We base discussions on the context taxonomy in Section VI on a concrete study of traceability recovery. This section describes the definition, design and settings of the experimentation, organized into the four phases *definition*, *planning*, *realization* and *interpretation* as specified in the experimental framework by Huffman Hayes and Dekhtyar [21]. Also, our work followed the general experimental guidelines by Wohlin *et al.* [48]. According to the proposed context taxonomy in Figure 2, our experiment is an evaluation conducted in the retrieval context, i.e., in the cave, using datasets from two industrial contexts.

A. Phase I: Definition

The definition phase presents the scope of the experimentation and describes the context. We entitle the study a *quasi-experiment* since there is no randomization in the selection of data sets nor IR tools.

1) Experiment Definition: The *goal* of the quasi-experiment is to evaluate traceability recovery tools in the context of embedded development, with the *purpose* of reporting results using proprietary software artifacts. The *quality focus* is to

Evaluation Context	Description	Evaluation methodology	Example measures
Level 4: Project context	Evaluations in a socio-organizational context. The IR tool is studied when used by engineers within the full complexity of an in-vivo setting.	Case studies	Project metrics, tool usage
Level 3: Work task context	Humans complete real work tasks, but in an in-vitro setting. Goal of evaluation is to assess the casual effect of an IR tool when completing a task.	Controlled experiments, case studies	Work task results, time spent
Level 2: Seeking context	A seeking context with a focus on how the human finds relevant information among what was retrieved by the system.	Technology-oriented experiments	Usability, MAP, DCG, DiffAR, Lag
Level 1: Retrieval context	A strict retrieval context, performance is evaluated wrt. the accuracy of a set of search results.	Benchmarks	Precision, recall, F-measure

TABLE I
FOUR INTEGRATED LEVELS OF CONTEXT IN IR-BASED TRACEABILITY RECOVERY EVALUATIONS.

evaluate precision and recall of tools from the *perspective* of a researcher who wants to evaluate how existing approaches to traceability recovery perform in a specific industrial context.

2) Industrial Context: The software artifacts in the industrial dataset are collected from a large multinational company active in the power and automation sector. The context of the specific development organization within the company is safety critical embedded development in the domain of industrial control systems. 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 [25], corresponding to a risk reduction factor of 1,000,000-10,000,000 for continuous operation. There are process requirements on maintenance of traceability information, especially between requirements and test cases. The software developers regularly perform tasks requiring traceability information, for instance when performing change impact analysis. Requirements and tests are predominantly specified in English NL text.

3) Characterization of Datasets: The first dataset used in our experiment originates from a project in the NASA Metrics Data Program, publicly available at COEST as the CM-1 dataset. The dataset specifies parts of a data processing unit and consists of 235 high-level requirements and 220 corresponding low-level requirements specifying detailed design. 361 traceability links between the requirement abstraction levels have been manually verified, out of 51 700 possible links. Items in the dataset have links to zero, one or many other items. This dataset is a de-facto benchmark of IR-based traceability recovery [5], thus we conduct a replication of previous evaluations.

The second dataset, referred to as the industrial data, consists of 225 requirements describing detailed design. These requirements are verified by 220 corresponding test case descriptions. The golden standard of 225 links was provided by the company, containing one link per requirement to a specific test case description. Thus, the link structure is different to the NASA data. The total number of combinatorial links is 49 500.

Both the NASA and the industrial datasets are bipartite, i.e., there exist only links between two subsets of software artifacts. Descriptive statistics of the datasets, calculated using the *Advanced Text Analyzer* at UsingEnglish.com[2], are presented in Tables II and III. Calculating *Gunning Fog Index* [19] as a complexity metric for requirement specifications written in English has been proposed by Farbey [17]. The second complexity metric reported in Tables II and III is the *Flesch Reading Ease*. Wilson *et al.* have previously calculated and reported it for requirement specifications from NASA [47]. Both datasets are considered large according to the framework of Huffman Hayes and Dekhtyar [21], even though we would prefer to label them very small.

Number of traceability links: 225		
Characteristic	Requirements	Test Case Descriptions
Items	224	218
Words	4 813	6 961
Words/Item	21.5	31.9
Avg. word length	6.5	7.0
Unique words	817	850
Gunning Fog Index	10.7	14.2
Flesch Reading Ease	33.7	14.8

TABLE II
DESCRIPTIVE STATISTICS OF THE INDUSTRIAL DATA

Number of traceability links: 361		
Characteristic	High-level Reqs.	Low-level Reqs.
Items	235	220
Words	5 343	17 448
Words/Items	22.7	79.3
Avg. word length	5.2	5.1
Unique words	1 056	2 314
Gunning Fog Index	7.5	10.9
Flesch Reading Ease	67.3	59.6

TABLE III
DESCRIPTIVE STATISTICS OF THE NASA DATA

B. Phase II: Planning

This section describes the traceability recovery tools and the experimental design.

1) Description of Tools: We selected two IR-based traceability recovery tools for our experiment. Requirements Tracing on Target (RETRO) was downloaded from the library of Open Source Software from NASA's Goddard Space Flight Center[3], however only binaries were available. Source code and binaries of ReqSimile was downloaded from the source code repository SourceForge[4].

[2]http://www.usingenglish.com/members/text-analysis/

[3]http://opensource.gsfc.nasa.gov/projects/RETRO/

[4]http://reqsimile.sourceforge.net/

RETRO, developed by Huffman Hayes *et al.*, is a tool that supports software development by tracing textual software engineering artifacts [23]. The tool we used implements VSM with features having term frequency-inverse document frequency weights. Similarities are calculated as the cosine of the angle between feature vectors [3]. RETRO also implements a probabilistic retrieval model. Furthermore, the tool supports relevance feedback from users using the Standard Rochio feedback. Stemming is done as a preprocessing step and stop word removal is optional according to the settings in the tool. Later versions of RETRO also implement LSI, but were not available for our experiments. We used RETRO version V.BETA, Release Date February 23, 2006.

ReqSimile, developed by Natt och Dag *et al.*, is a tool with the primary purpose to provide semi-automatic support to requirements management activities that rely on finding semantically similar artifacts [35]. Examples of such activities are traceability recovery and duplicate detection. The tool was intended to support the dynamic nature of market-driven requirements engineering. ReqSimile also implements VSM and cosine similarities. An important difference to RETRO is the feature weighting; terms are weighted as $1 + log_2(freq)$ and no inverse document frequencies are considered. Stop word removal and stemming is done as preprocessing steps. In our experimentation, we used version 1.2 of ReqSimile.

To get yet another benchmark for comparisons, a tool was implemented using a naïve tracing approach as suggested by Menzies *et al.* [32]. The *Naïve* tool calculates the number of terms shared by different artifacts, and produces ranked lists of candidate links accordingly. This process was done without any stop word removal or stemming.

2) Experimental Design: The quasi-experiment has two independent variables: the IR-based traceability recovery tool used and the input dataset. The first one has three factors: RETRO, ReqSimile and Naïve as explained in Section IV-B1. The second independent variable has two factors: Industrial and NASA as described in Section IV-A3. Consequently, six test runs were required to get a full factorial design.

IR-based traceability recovery tools are in the innermost evaluation context evaluated by verifying how many suggested links above a certain similarity threshold are correct, compared to a hand-crafted gold standard of correct links. Then, as presented in Figure 2, the laboratory model of IR evaluation is applied, thus recall and precision constitute our dependent variables. Recall and precision measure both the percentage of correct links recovered by a tool, and the amount of false positives. The aggregate measure F-score was also calculated, defined as the harmonic mean of precision and recall [3]:

$$F = 2 * \frac{precision * recall}{precision + recall}$$

Our null hypotheses, guiding the data analysis, are stated below. Performance is considered wrt. recall and precision.

NH1 The two tools implementing the vector space model, RETRO and ReqSimile, show equivalent performance.

NH2 Performance differences between the tools show equivalent patterns on the NASA and Industrial datasets.

NH3 RETRO and ReqSimile do not perform better than the Naïve tool.

C. Phase III: Realization

This section describes how the data was converted to valid input formats, and the actual tool usage.

1) Preprocessing of Datasets: Preprocessing the datasets was required since RETRO and ReqSimile use different input formats. The NASA dataset was available in a clean textual format and could easily be converted. The industrial data was collected as structured Microsoft Word documents including references, diagrams, revision history etc. We manually extracted the textual content and removed all formatting.

2) Experimental Procedure: Conducting the experiment consisted of six test runs combining all tools and datasets. In RETRO, two separate projects were created and the tool was run using default settings. We did not have access to neither a domain specific thesaurus nor a list of stop words. Relevance feedback using the standard Rochio method was not used. The *Trace All* command was given to calculate all candidate links, no filtering was done in the tool.

ReqSimile does not offer any configuration of the underlying IR models. The two input datasets were separated in two projects. After configuring the drivers of the database connections, the commands *Fetch requirements* and *Preprocess requirements* were given and the lists of candidate links were presented in the *Candidate requirements* tab.

The Naïve tool uses the same input format as RETRO. The tool does not have a graphical user interface, and was executed from a command-line interface.

V. Results and Interpretation

This section presents the results from our six test runs.

A. Phase IV: Interpretation

Huffman Hayes and Dekhtyar define the interpretation context as "the environment/circumstances that must be considered when interpreting the results of an experiment" [21]. We conduct our evaluation in the retrieval context as described in Section III. Due to the small number of datasets studied, our hypotheses are not studied in a strict statistical context.

The precision-recall graphs and the plotted F-scores are used as the basis for our comparisons. All hypotheses do to some extent concern the concept of equivalence, which we study qualitatively in the resulting graphs. However, presenting more search results than a user would normally consider adds no value to a tool. We focus on the top ten search results, in line with recommendations from previous research [33], [46], [43], and common practise in web search engines. The stars in Figures 3 and 4 indicate candidate link lists of length 10.

The first null hypothesis stated that the two tools implementing the VSM show equivalent performance. Figures 3 and 5 show that RETRO and ReqSimile produce candidate links

of equivalent quality, the stars are even partly overlapping. However, Figures 4 and 6 show that RETRO outperforms ReqSimile on the NASA dataset. As a result, the first hypothesis is rejected; *the two IR-based traceability recovery tools RETRO and ReqSimile, both implementing VSM, do not perform equivalently*.

The second null hypothesis stated that performance differences between the tools show equivalent patterns on the both datasets. The first ten datapoints of the precision-recall graphs, representing search hits of candidate links with lengths from 1 to 10, show linear quality decreases for both datasets. Graphs for the industrial data starts with higher recall values for short candidate lists, but drops faster to precision values of 5% compared to the NASA data. The Naïve tool performs better on the industrial data than on the NASA data, and the recall values increase at a higher pace, passing 50% at candidate link lists of length 10. The second hypothesis is rejected; *the tools show different patterns on the industrial dataset and the NASA dataset*.

The third null hypothesis, RETRO and ReqSimile do not perform better than the Naïve tool, is also rejected. Our results show that the Naïve tool, just comparing terms without any preprocessing, does not reach the recall and precision of the traceability recovery tools implementing VSM. *RETRO and ReqSimile perform better than the Naïve tool*.

VI. Discussion

In this section, the results from the quasi-experiment and related threats to validity are discussed. Furthermore, we discuss how we could conduct evaluations in outer contextual levels based on this study, and we discuss how to advance evaluations of IR-based traceability recovery in general.

A. Implication of Results

The IR-based traceability recovery tools RETRO and ReqSimile perform equivalently on the industrial dataset and similarly on the NASA data. From reading documentation and code of RETRO and ReqSimile, it was found that the tools construct different feature vectors. RETRO, but not ReqSimile, takes the inverse document frequency of terms into account when calculating feature weights. Consequently, terms overly frequent in the document set are not down-weighted as much in ReqSimile as in RETRO. This might be a major reason why RETRO generally performs better than ReqSimile in our quasi-experiment, even without the use of optional stop word removal. This shows that the construction of feature vectors is important to report when classifying traceability recovery tools, an aspect that often is omitted when reporting overviews of the field.

Our experimentation was conducted on two bipartite datasets of different nature. The NASA data has a higher density of traceability links and also a more complex link structure. RETRO and ReqSimile both perform better on the industrial dataset. The average amount of words of this dataset is fewer than in the NASA dataset, the reason for better IR performance is rather the less complex link structure. Not surprisingly, the performance of the traceability recovery is heavily dependant on the dataset used as input. Before there is a general large-scale dataset available for benchmarking, traceability recovery research would benefit from understanding various types of software artifacts. Especially for proprietary datasets used in experiments, characterization of both industrial context and the dataset itself must be given proper attention.

As mentioned in Section I, our quasi-experiment is partly a replication of studies conducted by Sundaram *et al.* [44], and Dekhtyar *et al.* [14]. Our results of using RETRO on the NASA dataset are similar, but not identical. Most likely, we have not applied the same version of the tool. Implementation of IR solutions forces developers to make numerous minor design decisions, i.e., details of the preprocessing steps, order of computations, numerical precision etc. Such minor variations can cause the differences in tool output we observe, thus version control of tools is important and should be reported.

B. Validity Threats

This section contains a discussion on validity threats to help define the creditability of the conclusions [48]. We focus on construct, internal and external validity.

Threats to *construct validity* concern the relationship between theory and observation. Tracing errors include both errors of inclusion and errors of exclusion. By measuring both recall and precision, the retrieval performance of a tool is well measured. However, the simplifications of the laboratory model of IR evaluation has been challenged [27]. There is a threat that recall and precision are not efficient measures of the overall usefulness of traceability tools. The question remains whether the performance differences, when put in a context with a user and a task, will have any practical significance. However, we have conducted a pilot study on RETRO and ReqSimile on a subset of the NASA dataset to explore this matter, and the results suggest that subjects supported by a slightly better tool also produce slightly better output [4].

Threats to *internal validity* can affect the independent variable without the researcher's knowledge and threat the conclusion about causal relationships between treatment and outcome. The first major threat comes from the manual preprocessing of data, which might introduce errors. Another threat is that the downloaded traceability recovery tools were incorrectly used. This threat was addressed by reading associated user documentation and running pilot runs on smaller dataset previously used in our department.

External validity concerns the ability to generalize from the findings. The bipartite datasets are not comparable to a full-size industrial documentation space and the scalability of the approach is not fully explored. However, a documentation space might be divided into smaller parts by filtering artifacts by system module, type, development team etc., thus also smaller datasets are interesting to study.

On the other hand, there is a risk that the industrial dataset we collected is a very special case, and that the impact of datasets on the performance of traceability recovery tools

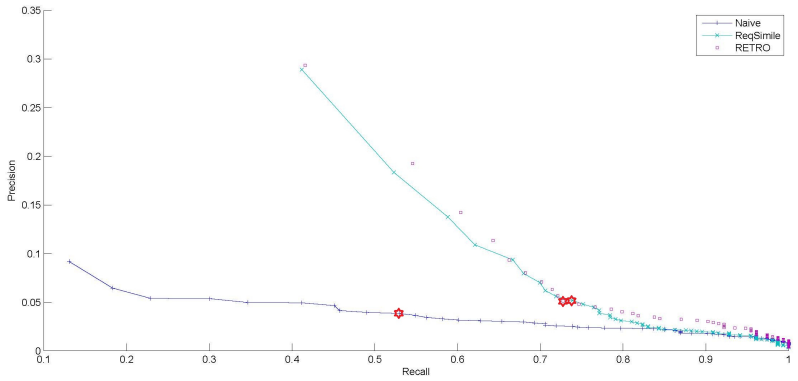

Fig. 3. Precision-recall graph for the Industrial dataset. The stars show candidate link lists of length 10.

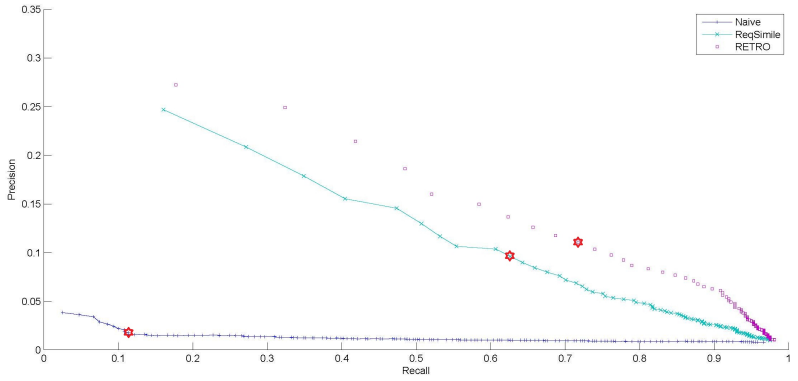

Fig. 4. Precision-recall graph for the NASA dataset. The stars show candidate link lists of length 10.

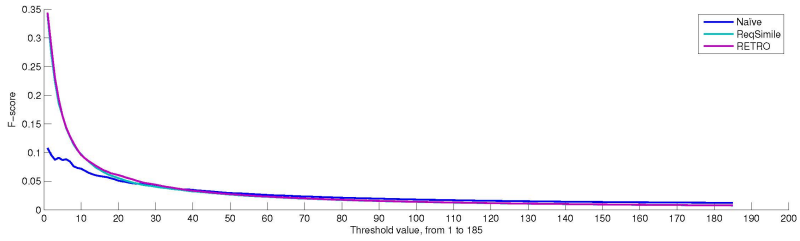

Fig. 5. F-Score for the Industrial dataset. The X-axis shows the length of candidate link lists considered.

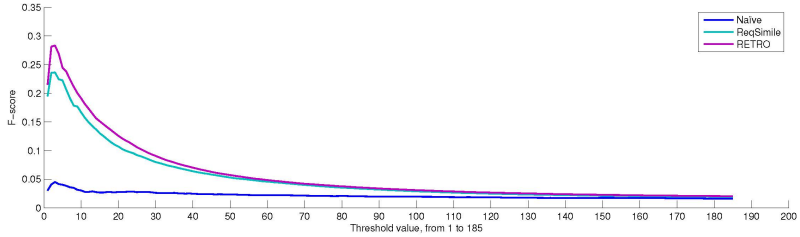

Fig. 6. F-Score for the NASA dataset. The X-axis shows the length of candidate link lists considered.

normally is much less. The specific dataset was selected in discussion with the company, to be representative and match our requirements on size and understandability. It could also be the case that the NASA dataset is not representative to compare RETRO and ReqSimile. The NASA data has been used in controlled experiments of RETRO before, and the tool might be fine-tuned to this specific dataset. Consequently, RETRO and ReqSimile must be compared on more datasets to enable firm conclusions.

C. Advancing to outer levels

The evaluation we have conducted resides in the innermost retrieval context of the taxonomy described in Section III. Thus, by following the experimental framework by Huffman Hayes *et al.* [21], and by using proprietary software artifacts as input, our contribution of empirical evidence can be classified as a Level 1 evaluation in an industrial environment, as presented in Figure 7. By adhering to the experimental framework, we provided enough level of detail in the reporting to enable future secondary studies to utilize our results.

Building upon our experiences from the quasi-experiment, we outline a possible research agenda to move the empirical evaluations in a more industry relevant direction. Based on our conducted Level 1 study, we could advance to outer levels of the context taxonomy. Primarily, we need to go beyond precision-recall graphs, i.e., step out of "the cave of IR evaluation". For example, we could introduce DCG as a secondary measure to analyze how the traceability recovery tools support finding relevant information among retrieved candidate links, repositioning our study as path A shows in the Figure 7.

However, our intention is to study how software engineers interact with the output from IR-based traceability recovery tools, in line with what we initially have explored in a pilot study [4]. Based on our experimental experiences, a future controlled experiment should be conducted with more subjects, and preferably not only students. An option would be to construct a realistic work task, using the industrial dataset as input, and run the experiment in a classroom setting. Such a research design could move a study as indicated by path B in Figure 7. Finally, to reach the outermost evaluation context as path C shows, we would need to study a real project with real engineers, or possibly settle for a student project. An option would be to study the information seeking involved in the state-of-practice change impact analysis process at the company from where the industrial dataset originates. The impact analysis work task involves traceability recovery, but currently the software engineers have to complete it without dedicated tool support.

D. Advancing traceability recovery evaluations in general

Our experiences from applying the experimental framework proposed by Huffman Hayes and Dekhtyar [21] are positive. The framework provided structure to the experiment design activity, and also it encouraged detailed reporting. As a result, it supports comparisons between experimental results, replications of reported experiments, and it supports secondary studies to aggregate empirical evidence. However, as requirements tracing constitutes an IR problem (for a given artifact, relations to others are to be identified), it must be evaluated according to the context of the user as argued by Ingwersen and Järvelin [24]. The experimental framework includes "interpretation context", but it does not cover this aspect of IR evaluation. Consequently, we claim that our context taxonomy fills a purpose, as a complement to the more practical experimental guidelines offered by Huffman Hayes and Dekhtyar's framework [21].

While real-life proprietary artifacts are advantageous for the relevance of the research, the disadvantage is the lack of accessibility for validation and replication purposes. Open source artifacts offer in that sense a better option for advancing the research. However, there are two important aspects to consider. Firstly, open source development models tend to be different compared to proprietary development. For example, wikis and change request databases are more important than requirements documents or databases [41]. Secondly, there are large variations *within* open source software contexts, as there is within proprietary contexts. Hence, it is critical that research matches pairs of open source and proprietary software, as proposed by Robinson and Francis [38], based on several characteristics, and not only their being open source or proprietary. This also holds for generalization from studies from one domain to the other, as depicted in Figure 7.

Despite the context being critical, also evaluations in the innermost evaluation context can advance IR-based traceability recovery research, in line with the benchmarking discussions by Runeson *et al.* [39] and suggestions by members of the COEST [12], [13], [7]. Runeson *et al.* refer to the automotive industry, and argue that even though benchmarks of crash resistance are not representative to all types of accidents, there is no doubt that such tests have been a driving force in making cars safer. The same is true for the TREC conferences as mentioned in Section I. Thus, the traceability community should focus on finding a series of meaningful benchmarks, including contextual information, rather than striving to collect a single large set of software artifacts to "rule them all". Regarding size however, such benchmarks should be considerably larger that the de-facto benchmarks used today. The same benchmark discussion is active within the research community on enterprise search, where it has been proposed to extract documents from companies that no longer exist, e.g., Enron [20], an option that might be possible also in software engineering.

Runeson *et al.* argue that a benchmark should not aim at statistical generalization, but a qualitative method of analytical generalization. Falessi *et al.* on the other hand, bring attention to the value of statistical hypothesis testing of tool output [16]. They reported a technology-oriented experiment in the seeking context (including secondary measures), and presented experimental guidelines in the form of seven empirical principles. However, the principles they proposed focus on the innermost contexts of the taxonomy in Figure 2, i.e., evaluations with-

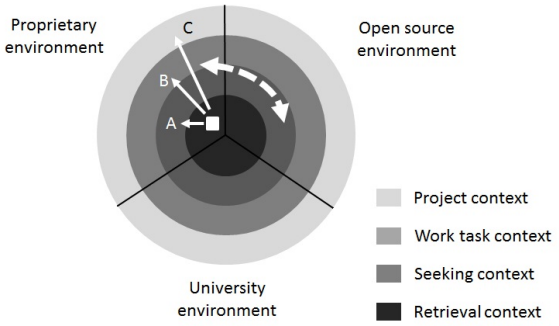

Fig. 7. Our quasi-experiment, represented by a square, mapped to the taxonomy. Paths A-C show options to advance towards outer evaluation contexts, while the dashed arrow represents the possibility to generalize between environments as discussed by Robinson and Francis [38].

out human subjects. Also, since the independence between datapoints on a precision-recall curve for a specific dataset is questionable, we argue that the result from each dataset instead should be treated as a single datapoint, rather than applying the cross-validation approach proposed by Falessi *et al.* As we see it, statistical analysis turns meaningful in the innermost evaluation contexts when we have access to sufficient numbers of independent datasets. On the other hand, when conducting studies on human subjects, stochastic variables are inevitably introduced, making statistical methods necessary tools.

Research on traceability recovery has the last decade, with a number of exceptions, focused more on tool improvements and less on sound empirical evaluations [5]. Since several studies suggest that further modifications of IR-based traceability recovery tools will only result in minor improvements [36], [45], [15], the vital next step is instead to assess the applicability of the IR approach in an industrial setting. The strongest empirical evidence on the usefulness of IR-based traceability recovery tools comes from a series of controlled experiments in the work task context, dominated by studies using student subjects [35], [23], [9], [4]. Consequently, to strengthen empirical evaluations of IR-based traceability recovery, we argue that contributions must be made along two fronts. Primarily, in-vivo evaluations should be conducted, i.e., industrial case studies in a project context. In-vivo studies on the general feasibility of the IR-based approach are conspicuously absent despite more than a decade of research. Thenceforth, meaningful benchmarks to advance evaluations in the two innermost evaluation contexts should be collected by the traceability community.

VII. Conclusions and Future Work

We propose a context taxonomy for evaluations of IR-based traceability recovery, consisting of four integrated levels of evaluation contexts (retrieval, seeking, work task, and project context), and an orthogonal dimension of study environments (university, open source, proprietary environment). To illustrate our taxonomy, we conducted an evaluation of the framework for requirements tracing experiments by Huffman Hayes and Dekhtyar [21].

Adhering to the framework, we conducted a quasi-experiment with two tools implementing VSM, RETRO and ReqSimile, on proprietary software artifacts from two embedded development projects. The results from the experiment show that the tools performed equivalently on the dataset with a low density of traceability links. However, on the dataset with a more complex link structure, RETRO outperformed ReqSimile. An important difference between the tools is that RETRO takes the inverse document frequency of terms into account when representing artifacts as feature vectors. We suggest that information about feature vectors should get more attention when classifying IR-based traceability recovery tools in the future, as well as version control of the tools. Furthermore, our research confirms that input software artifacts is an important factor in traceability experiments. Research on traceability recovery should focus on exploring different industrial contexts and characterize the data in detail, since replications of experiments on closed data are unlikely.

Following the experimental framework supported our study by providing structure and practical guidelines. However, it lacks a discussion on the evaluation contexts highlighted by our context taxonomy. On the other hand, when combined, the experimental framework and the context taxonomy offer a valuable platform, both for conducting and discussing, evaluations of IR-based traceability recovery.

As identified by other researchers, the widely used measures recall and precision are not enough to compare the results from tracing experiments [22]. The laboratory model of IR evaluation has been questioned for its lack of realism, based on progress in research on the concept of relevance and information seeking [27]. Critics claim that real human users of IR systems introduce non-binary, subjective and dynamic relevance, which affect the overall IR process. Our hope is that our proposed context taxonomy can be used to direct studies beyond "the cave" of IR evaluation, and motivate more industrial case studies in the future.

Acknowledgement

This work was funded by the Industrial Excellence Center EASE – Embedded Applications Software Engineering[5]. Special thanks go to the company providing the proprietary dataset.

References

[1] G. Antoniol, G. Canfora, A. De Lucia, and G. Casazza. Information retrieval models for recovering traceability links between code and documentation. In *Proceedings of the International Conference on Software Maintenance*, page 40, 2000.

[2] G. Antoniol, G. Canfora, A. De Lucia, and E. Merlo. Recovering code to documentation links in OO systems. In *Proceedings of the 6th Working Conference on Reverse Engineering*, pages 136–44, 1999.

[3] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[4] M. Borg and D. Pfahl. Do better IR tools improve the accuracy of engineers' traceability recovery? In *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*, pages 27–34, 2011.

[5] http://ease.cs.lth.se

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

[6] E. Charrada, D. Caspar, C. Jeanneret, and M. Glinz. Towards a benchmark for traceability. In *Proceedings of the 12th International Workshop on Principles on Software Evolution and 7th ERCIM Workshop on Software Evolution*, 2011.

[7] J. Cleland-Huang, A. Czauderna, A. Dekhtyar, O. Gotel, J. Huffman Hayes, E. Keenan, J. Maletic, D. Poshyvanyk, Y. Shin, A. Zisman, G. Antoniol, B. Berenbach, A. Egyed, and P. Maeder. Grand challenges, benchmarks, and TraceLab: developing infrastructure for the software traceability research community. In *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, 2011.

[8] C. Cleverdon. The significance of the cranfield tests on index languages. In *Proceedings of the 14th Annual International SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12, 1991.

[9] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *Transactions on Software Engineering Methodology*, 16(4):13, 2007.

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

[11] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[12] A. Dekhtyar and J. Huffman Hayes. Good benchmarks are hard to find: Toward the benchmark for information retrieval applications in software engineering. *Proceedings of the 22nd International Conference on Software Maintenance*, 2006.

[13] A. Dekhtyar, J. Huffman Hayes, and G. Antoniol. Benchmarks for traceability? In *Proceedings of the International Symposium on Grand Challenges in Traceability*, 2007.

[14] A. Dekhtyar, J. Huffman Hayes, and J. Larsen. Make the most of your time: How should the analyst work with automated traceability tools? In *Proceedings of the 3rd International Workshop on Predictor Models in Software Engineering*, 2007.

[15] D. Falessi, G. Cantone, and G. Canfora. A comprehensive characterization of NLP techniques for identifying equivalent requirements. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, pages 1–10, 2010.

[16] D. Falessi, G. Cantone, and G. Canfora. Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques. *Transactions on Software Engineering*, 99(1), 2011.

[17] B. Farbey. Software quality metrics: considerations about requirements and requirement specifications. *Information and Software Technology*, 32(1):60–64, 1990.

[18] O. Gotel and A. Finkelstein. An analysis of the requirements traceability problem. Technical report, Department of Computing, Imperial College, 1994.

[19] R. Gunning. *Technique of clear writing; rev. version*. McGraw-Hill, 1968.

[20] D. Hawking. Challenges in enterprise search. In *Proceedings of the 15th Australasian database conference*, pages 15–24, 2004.

[21] J. Huffman Hayes and A. Dekhtyar. A framework for comparing requirements tracing experiments. *International Journal of Software Engineering and Knowledge Engineering*, 15(5):751–781, 2005.

[22] J. Huffman Hayes, A. Dekhtyar, and S. Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *Transactions on Software Engineering*, 32:4–19, 2006.

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

[24] P. Ingwersen and K. Järvelin. *The Turn: Integration of Information Seeking and Retrieval in Context*. Springer, 2005.

[25] International Electrotechnical Commission. IEC 61508, Functional Safety of Electrical/Electronic/Programmable Electronic Safety Related Systems - Part 3: Software requirements. *65A/550/FDIS*, 2009.

[26] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 41–48. ACM, 2000.

[27] J. Kekäläinen and K. Järvelin. Evaluating information retrieval systems under the challenges of interaction and multidimensional dynamic relevance. In *Proceedings of the CoLIS 4 Conference*, pages 253–270, 2002.

[28] J. Lin, C. Lin, J. Cleland-Huang, R. Settimi, J. Amaya, G. Bedford, B. Berenbach, O. BenKhadra, C. Duan, and X. Zou. Poirot: A distributed tool supporting enterprise-wide automated traceability. In *Proceedings of the International Conference on Requirements Engineering*, pages 356–357, 2006.

[29] M. Lormans and A. van Deursen. Can LSI help reconstructing requirements traceability in design and test? In *Proceedings of the 10th European Conference on Software Maintenance and Reengineering*, pages 47–56, 2006.

[30] C. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[31] A. Marcus and J. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proceedings of the 25th International Conference on Software Engineering*, pages 125–135, 2003.

[32] T. Menzies, D. Owen, and J. Richardson. The strangest thing about software. *Computer*, 40:54–60, 2007.

[33] G. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63:81–97, 1956.

[34] P. Morville. *Ambient Findability: What We Find Changes Who We Become*. O'Reilly Media, 2005.

[35] J. Natt Och Dag, T. Thelin, and B. Regnell. An experiment on linguistic tool support for consolidation of requirements from multiple sources in market-driven product development. *Empirical Software Engineering*, 11:303–329, 2006.

[36] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia. On the equivalence of information retrieval methods for automated traceability link recovery. In *Proceedings of the 18th International Conference on Program Comprehension*, pages 68–71, 2010.

[37] S. E. Robertson and S. Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3):129–146, 1976.

[38] B. Robinson and P. Francis. Improving industrial adoption of software engineering research: a comparison of open and closed source software. In *Proceedings of the 4th International Symposium on Empirical Software Engineering and Measurement*, pages 21:1–21:10, 2010.

[39] P. Runeson, M. Skoglund, and E. Engström. Test benchmarks - what is the question? In *International Conference on Software Testing Verification and Validation Workshop*, pages 368–371, April 2008.

[40] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[41] W. Scacchi. Understanding the requirements for developing open source software systems. *IEEE Software*, 149(1):24–39, 2002.

[42] A. Smeaton and D. Harman. The TREC experiments and their impact on Europe. *Journal of Information Science*, 23(2):169–174, 1997.

[43] K. Sparck Jones, S. Walker, and S. E. Robertson. A probabilistic model of information retrieval: development and comparative experiments. *Information Processing and Management*, 36(6):779–808, 2000.

[44] S. Sundaram, J. Huffman Hayes, and A. Dekhtyar. Baselines in requirements tracing. In *Proceedings of the Workshop on Predictor Models in Software Engineering*, pages 1–6, 2005.

[45] S. Sundaram, J. Huffman Hayes, A. Dekhtyar, and A. Holbrook. Assessing traceability of software engineering artifacts. *Requirements Engineering*, 15(3):313–335, 2010.

[46] T. Welsh, K. Murphy, T. Duffy, and D. Goodrum. Accessing elaborations on core information in a hypermedia environment. *Educational Technology Research and Development*, 41:19–34, 1993.

[47] W. Wilson, L. Rosenberg, and L. Hyatt. Automated analysis of requirement specifications. In *Proceedings of the International Conference on Software Engineering*, pages 161–171, 1997.

[48] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, 2000.