SEMANTICALLY ENHANCED TRACEABILITY ACROSS SOFTWARE AND SYSTEM-RELATED NATURAL LANGUAGE ARTIFACTS

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Jin L.C. Guo

Jane Cleland-Huang, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana
October 2017

SEMANTICALLY ENHANCED TRACEABILITY ACROSS SOFTWARE AND SYSTEM-RELATED NATURAL LANGUAGE ARTIFACTS

Abstract

by

Jin L.C. Guo

This dissertation focuses on accurate trace link creation for software projects. Trace links represent established associations between software requirements, design, code, test cases and other such artifacts. Traceability describes the potential to create and maintain trace links during the software development life cycle. In most safety-critical domains, the need for traceability is prescribed by certifying bodies.

Creating trace links manually is time consuming and error prone. Automated solutions use information retrieval and machine learning techniques to generate trace links; however, current techniques fail to understand semantics of the software artifacts or to integrate domain knowledge into the tracing process and therefore tend to deliver imprecise and inaccurate results. Therefore, this dissertation proposes a series of traceability solutions with different semantic enhancement strategies that aim at improving the quality of trace link generation between regulations, software requirements and design documents written in natural language. The first approach augments software artifacts with an ontology of domain terms and relations. To further increase the trace link accuracy, an intelligent tracing system DoCIT is proposed that is able to reason over artifact semantics through use of a domain ontology and a set of trace heuristics. Finally, a deep learning based tracing method is presented that represents and compares artifact semantics in an implicit but fully automated way.

The main contribution of this dissertation is in addressing the lack of semantic knowledge in current traceability solutions by developing techniques which extract semantic knowledge, integrate it into the tracing process, and thereby deliver more accurate and trustworthy traceability solutions.

CONTENTS

FIGURES

TABLES

ACKNOWLEDGMENTS

I am deeply grateful to my advisor Jane Cleland-Huang for guiding me on how to become a Software Engineering researcher. In the past five years, I was constantly inspired by her knowledge and experience during our interactions. I am especially motivated by her passion for conducting novel research in the field of SE. She demonstrates to me how fun and rewarding the research experience could be. She is also very considerate and thoughtful about the challenges I have been through both in career and life, and offers me valuable advice. I couldn't have asked for a better advisor.

It is my great privilege to have Jane Hayes, Collin McMillan, and David Chiang in my committee. Jane Hayes is one of the leading researchers in the field of Software Traceability. I was fortunate enough to get her perspective on my work. As a brilliant SE researcher, Collin McMillan has given me many constructive and concrete suggestions on how to organize and improve my dissertation. David Chiang has vast knowledge on Natural Language Processing. His feedback was especially valuable and inspiring to me. I always received extraordinary kindness and support when I talked with everyone in the committee.

I would like to thank Prof. Gregory Madey, the Director of Graduate Studies in the Department of Computer Science and Engineering at the University of Notre Dame. He generously offers guidance and encouragement to all of us. Also many thanks to the administrative staff, especially Joyce Yeats, who always looks out for us and make sure that we are having a smooth graduate study.

Thanks to the National Science Foundation for funding this work (under Grant No. CCF-1319680) and supporting me to attend many conferences and events.

Personally, I greatly appreciate my family and friends who are either close by and far

away. They are always there when I seek for help.

Mostly I would like to thank my husband Jinghui Cheng, who is the ideal life companion to me and wonderful father to our children. His love and support give me the greatest strength I could ever get.

CHAPTER 1

INTRODUCTION

1.1 Traceability Overview

Software artifacts include all the primary and secondary data produced during the software development life cycle. The success of any software engineering project is dependent upon its software artifacts. Use cases, requirement specifications, design descriptions, source code, and many other artifacts are all generated as natural byproducts of the systems' development process. These data are not created independently. On the contrary, they are associated through specific semantic relationships. For example, one design element **fulfils** one requirement of the system, or one method defined in the source code **implements** one element of the design. Each association is called one "trace" or "trace link", as depicted in Figure 1.1, and contains one pair of source and target artifacts, and a specific trace link type (e.g. implements). The notion "Traceability" describes the potential for trace links to be established and used in software projects [49].

Software traceability is not achieved automatically. Essential activities are required to generate, maintain and use trace links throughout the software development process. These activities should be carefully planned and managed as a traceability strategy. Figure 1.2 illustrates a generic traceability process model in which the main traceability activities are mapped onto the process along with each activity's responsibilities, inputs and outputs.

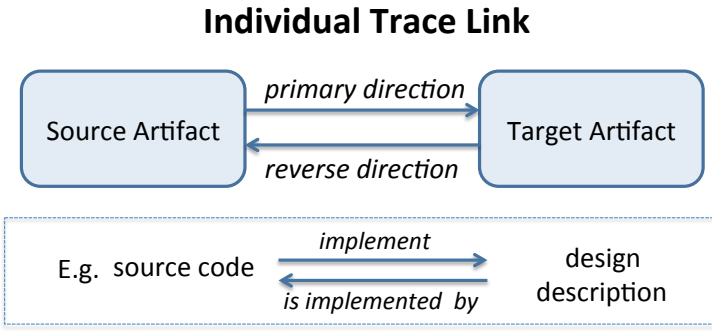


Figure 1.1. An individual trace link

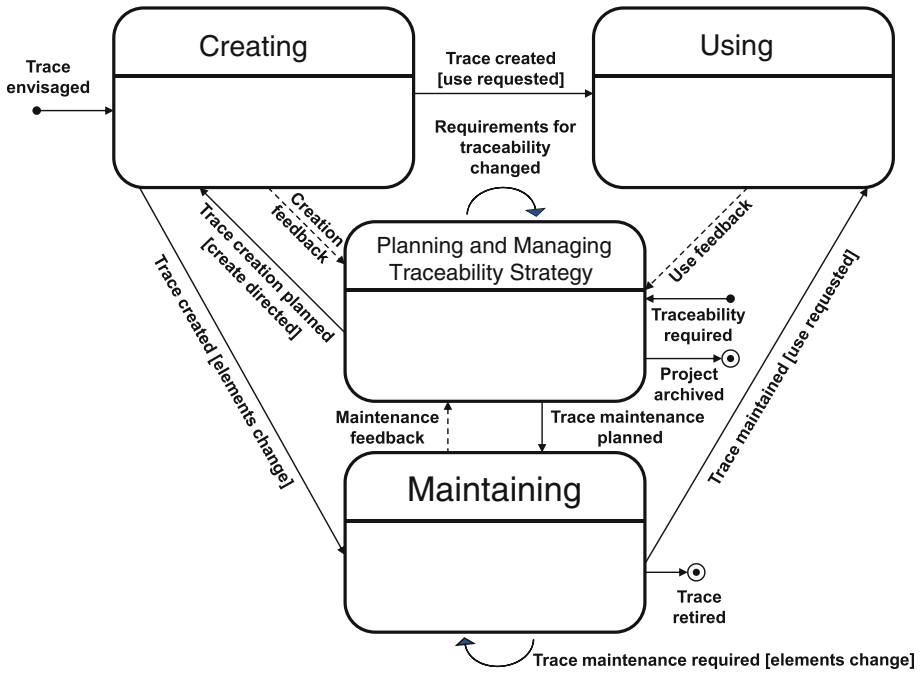


Figure 1.2. A generic Traceability Model [49].
Permission to reuse this figure has been granted by Springer eBook.

1.2 Challenge in Trace Link Generation

Software traceability is regarded as an important quality to aid in the evaluation of the software product and its engineering process. Especially for safety-critical systems, such as medical device production and aircraft operation control, traceability is required by regulatory authorities such as the USA Food and Drug Administration (FDA) and US Federal Aviation Authority (FAA) as an essential component for their approval and certification process [42, 121, 79]. In order to demonstrate that such systems are safe for use, trace links need to be established and maintained between unique software artifacts including hazards, faults, requirements, design, code, test cases, etc. Furthermore, the connections between software artifacts established by trace links can effectively support a variety of software engineering activities, such as impact analysis, test regression selection, and coverage analysis [50, 88, 118].

Despite its importance and potential benefits, traceability is often neglected during the software development process and in many projects trace links are created in an ad hoc manner due to the massive human effort needed to create and maintain links [51]. Such practices inevitably result in deficient trace links. Recently, Mäder et al. conducted a detailed analysis of the traceability documents submitted to the FDA as part of the medical device approval process in the USA [107]. It revealed that trace links are often created immediately prior to certification and they tend to be inconsistent, incomplete, and even conflicting. A more strategic traceability solution is necessary to generate accurate trace links and furthermore to effectively utilize trace links for software engineering activities [33].

Manually creating software trace links is an arduous task, especially the trace links between artifacts that are written in natural language. Creating those trace links requires human analysts to first carefully read the content of the artifacts, to understand their semantic implications, and then to properly assess their relationship using relevant knowledge of the target system and the application domain. As the number of software artifacts increase,

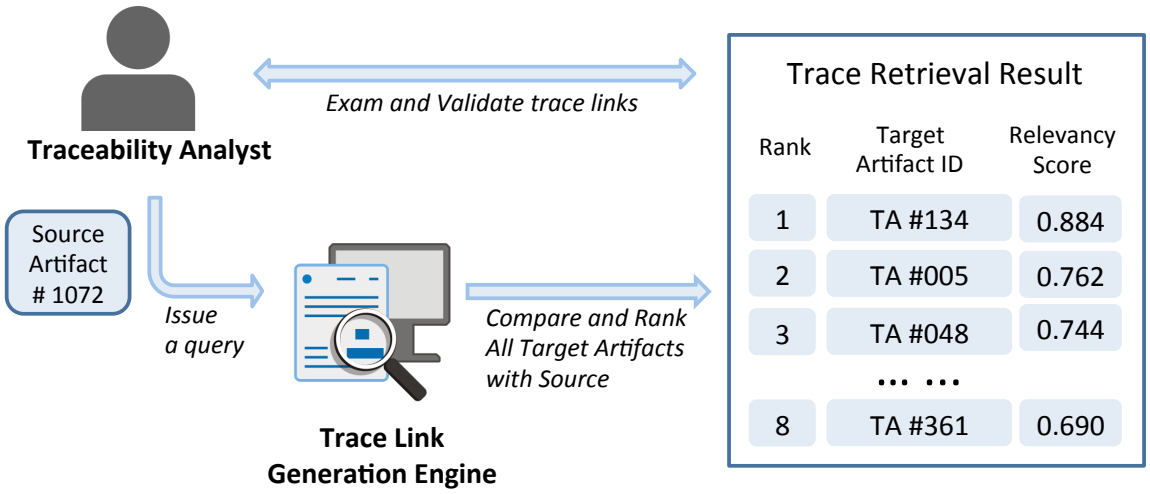


Figure 1.3. A typical trace retrieval process

the effort for generating their trace links grows exponentially. For example, Berenbach et al. described a 30,000 requirement project with 300 different sets of relevant regulatory codes [14]. The costs for establishing the trace links between requirements and their relevant regulatory codes were estimated by the projects requirements specialist at over USD$1.6 Million.

To lighten the burden of trace link creation, many automated or semi-automated approaches have been proposed to enable users to dynamically generate trace links between source and target artifacts with supporting tools. For example, a user might issue a specified requirement as a query (i.e. source), and expect those tools to retrieve all the design descriptions (i.e. targets) that are associated with this requirement. This process is illustrated in Figure 1.3. The retrieval method is normally based on comparing the textual content of artifacts using Information Retrieval (IR) algorithms [4]. For example, the most commonly used algorithm, the Vector Space Model (VSM), represents both source and target artifacts as vectors based on their contained terms and calculates the similarity of those vectors as the relevancy measure between source and target artifacts [65]. The Basic

VSM method has several drawbacks. For instance, it cannot effectively retrieve artifacts with inconsistent usage of terms. VSM integrated with user feedback or a thesaurus is capable of solving the term inconsistencies for some cases such as acronym and synonym occurrence [65]. Other state-of-the-art trace retrieval methods include Latent Semantic Indexing (LSI) or Latent Dirichlet Allocation (LDA) that analyze term distributions across a set of documents and discover latent topics for representing and comparing software artifacts [37, 6, 110]. Unfortunately, despite all these advances, the quality of retrieved trace links using these approaches is still far from satisfactory, i.e. the list of retrieved links contains many invalid links (artifacts are incorrectly linked to the source) while omitting many valid links [89]. Therefore, users cannot trust the retrieved results and they have to examine a long list of candidate links to confirm their validity. It is even more difficult for analysts to locate missing links that algorithms fail to retrieve [65], because finding these may require a search through the entire artifact set. As a result, the task of creating and maintaining accurate trace links in cost-effective ways is still regarded as one of the grand challenges for traceability research [48].

1.3 Towards Accurate Software Traceability

A closer examination of how human experts evaluate an actual trace link would explain why methods such as VSM and LSI fall short on this task. For example, one requirement for a health information system specified by the Certification Commission for Health Information Technology (CCHIT)[1] states that "When passwords are used, the system shall not transport passwords in plain text." This requirement is traced to a security standard defined by the Health Insurance Portability and Accountability Act (HIPAA)[2] written as "implement technical security measures to guard against unauthorized access to electronic

[1] https://www.cchit.org/

[2] http://www.hhs.gov/hipaa/

protected health information that is being transmitted over an electronic communications network." A human expert with understanding of the target system and its domain can establish the connection between these two artifacts with the following information: using a **password** is one way to achieve authentication when a person or entity seeks access to the **protected health information**; when a password is **transmitted** through open networks, it should be **encrypted** with defined encryption solutions rather than **plain text**. This trace link, however, is unlikely to be retrieved by any of the techniques introduced earlier. The vectors used in those approaches (either comprised of terms or topics) are not capable of representing the rich semantic content of the original software artifacts. Meanwhile, the simple similarity or relevancy measures calculated by those approaches fail to draw reasonable inference from the artifact semantics and domain knowledge. Given the wide semantic and knowledge gap between current trace link creation approaches and the manual trace link evaluation process, the work described in this dissertation aims at utilizing different techniques including natural language processing, knowledge extraction, and neural networks to fill the gap and thus to contribute towards meeting the grand challenge of accurate and trusted traceability.

1.4 Dissertation Outline

This Dissertation is comprised of published and in-progress work for semantically enhanced traceability solutions. We focus on a specific subset of artifact types that are represented using natural language including *regulations*, *requirements* and *design documents*. The trace links between other formatted or semi-formatted artifacts such as UML and source code are not in the scope of the work presented in this dissertation. The dissertation starts by describing work designed to improve trace accuracy using query modification techniques in Chapter 2, followed by an intelligent tracing system integrating domain knowledge in the format of ontology and a set of trace heuristic rules in Chapters 3 and 4. Chapter 5 illustrates a deep learning based approach for artifact semantic representation

and comparison that eliminates the dependence on manually crafted ontology and heuristics. In Chapter 6, we extend the deep learning based approach and apply it to different datasets. Finally, Chapter 7 concludes the dissertation and briefly describes future directions that might further improve the semantically enhanced solutions for the traceability problem. Figure 1.4 demonstrates the progress of each individual contribution towards the goal of semantically enhanced traceability. More details about the content from Chapter 2 to Chapter 6 are described below.

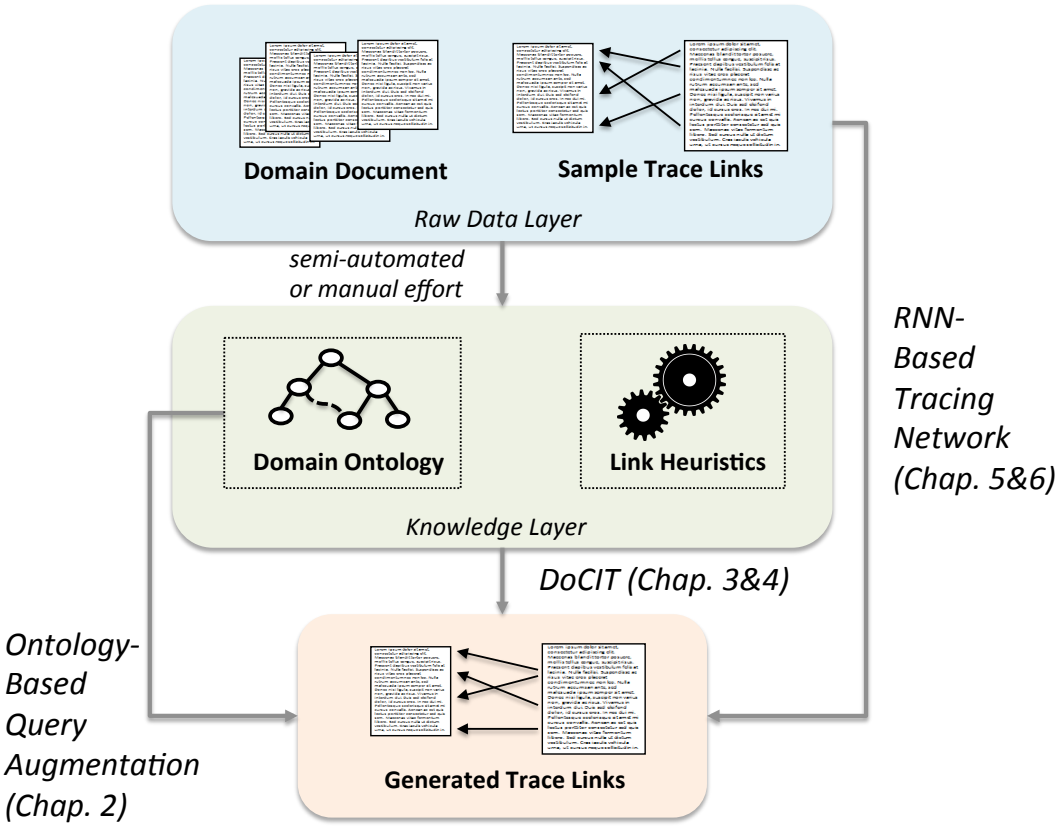


Figure 1.4. Progress towards semantically enhanced traceability solutions

Drawing an analogy between conventional Information Retrieval (IR) and trace link retrieval processes, each source artifact is considered as a query representing the user's information need, while all the target artifacts are considered as a collection of documents that potentially contain content to satisfy the user's need. The Vector Space Model (VSM) approach is easy to implement and fast to execute for IR tasks; but it suffers from two main weaknesses. Firstly, the inconsistency of term usage between source and target artifacts prevents useful documents from being retrieved. Meanwhile, terms interpreted out-of-context cause incorrect matches resulting in retrieving documents that are actually not useful. Query modification is an effective method to address both problems of VSM by augmenting the original queries with more "relevant" context terms [23]. Chapter 2 investigates the extent to which the trace link retrieval task benefits from query modification techniques and which query modification techniques lead to the greatest improvements. Specifically, it proposes a new technique that augments the tracing query with related terms appearing in a domain ontology. The ontology is constructed manually using a guided approach that leverages existing trace links. This chapter also introduces two other baseline techniques, i.e. the classification technique that trains a classifier to modify the original query (source artifact) with terms learned from a training set of trace links, and the web-mining technique that modifies the query by identifying high frequency and domain-specific terms from a set of domain documents mined from a web search. The performance of each method is evaluated on ten datasets traced to the same security regulations from the USA government's Health Insurance Privacy and Portability Act (HIPAA) and is compared against the basic VSM method. The classifier method yields the greatest improvement followed by the ontology-based query augmentation. However, creating the training set for the classifier requires significantly more person-hours than the ontology-based approach. This work was published in the *Journal of Empirical Software Engineering* [62].

Chapter 2 demonstrates that ontology-based query augmentation only achieved limited improvement over VSM. Although similar to an IR task, the trace link evaluation task also

has its own distinct characteristics. For example, its "query" is not a short list of individual terms. Rather, the tracing query is normally a coherent sentence extracted from a software artifact with complex semantics. The artifact semantics cannot be fully represented by a bag-of-words (or a bag of modified/augmented words). Therefore, Chapter 3 proposes a Domain-Contextualized Intelligent Traceability Solution (DoCIT) that can better represent and compare the artifact semantics using natural language processing methods and a domain ontology. DoCIT performs the trace link evaluation by first populating an "action frame" from a software artifact based on the artifact's syntax. It then compares action frames obtained from source and target artifacts using an ontology of domain terms and relations. The specific comparison strategies are defined by a set of link-creation heuristics. DoCIT is designed to approximate some of the higher level reasoning processes that a human expert uses to perform this task. It is demonstrated to significantly increase the trace link accuracy with a complete knowledge base of domain ontology and link heuristics. It also shows marked improvements when only a partial knowledge base is available. The evaluation is performed on a software-intensive system for Transportation Operation Control. The work described in this chapter was published in the *Proceedings of the International Conference on Automated Software Engineering* [58] and is based on an earlier paper published in the *Proceedings of the International Conference on Requirements Engineering* [57].

Most of the trace link creation methods can only return to the user a similarity or relevancy score for each link. This is not informative enough for a user to validate the links. Since DoCIT is able to capture the "essence" of artifacts and to perform logical reasoning over their relationship, we can maximize the information conveyed to the user in order to further facilitate the trace link creation task. Towards this direction, the work described in Chapter 4 enhances DoCIT with the function of automatically generating trace link rationales in natural language and the work was published in the *Proceedings of the International Conference on Requirements Engineering* [59]. By adding a set of

rationale generation patterns, DoCIT is able to explain the domain facts that appeared in the trace link and the specific relations between source and target artifact. This process is illustrated with examples of automatically generated rationales taken from domains of both Transportation Operation Control and Medical Infusion pump.

The DoCIT method described in Chapters 3 and 4 requires a considerable amount of manual effort to create domain ontology and link heuristics. In addition, DoCIT is sensitive to errors that the syntactic parser makes when it processes the artifacts. To overcome these weaknesses, Chapter 5 proposes a deep learning based tracing method that bypasses the dependence of explicit domain ontology and trace heuristics and the usage of a parser, and is therefore more scalable and robust. The proposed tracing network architecture utilizes word embedding technique and Recurrent Neural Network (RNN) models to automatically generate trace links. The Bidirectional Recurrent Gated Unit (BI-GRU) model in neural networks is especially effectively for constructing semantic associations between artifacts. It significantly improves the mean average precision of generated trace links in comparison with both VSM and LSI on a large industrial dataset. The current progress shows the potential of automating the creation of accurate trace links in industrial-strength datasets and this work was published in the *Proceedings of the International Conference on Software Engineering* [61].

In order to further improve the tracing network to better distinguish valid links from invalid ones, Chapter 6 investigates the negative link sampling technique. The result indicates that the tracing network could benefit from being exposed to more negative examples during the training process. On the other hand, for many software engineering projects, a large quantity of trace links are not available either due to the nature of the projects or the considerable manual effort needed to create those links. Therefore, to investigate if the tracing network can be used when only a limited number of trace links are available, in Chapter 6 we also apply the tracing network to two additional smaller sized datasets from the domains of Transportation Operation Control and Electronic Health Record. We

observe that the tracing network does not bring an obvious advantage over VSM when applied on its own; however, a hybrid approach that leverages both VSM and the tracing network is able to significantly increase the trace accuracy on both datasets. We conclude this chapter by discussing future directions that can potentially enhance the deep learning based methods and reach more generalizable tracing solutions.

CHAPTER 2

TACKLING THE TERM-MISMATCH PROBLEM IN AUTOMATED TRACE RETRIEVAL BY QUERY MODIFICATION

This chapter presents the work entitled "Tackling the Term-Mismatch Problem in Automated Trace Retrieval" that was published in the *Journal of Empirical Software Engineering (EMSE)* [62]. The content is reordered and refined to reflect my contribution which focuses on introducing an ontology-based approach – a means to utilize domain knowledge to semantically enhance tracing queries. The background is introduced first. The proposed ontology-based tracing technique is then described in detail followed by shorter descriptions of two other trace query modification techniques used as comparative baselines. These two query augmentation techniques were previously reported in [32] and [47]. The tradeoffs of all three techniques are discussed thoroughly at the end of this chapter.

2.1 Introduction

In this chapter, we focus on the non-trivial problem of automating traceability between regulatory codes and system level requirements. This represents a particularly compelling and challenging problem because industries are often required by certifying bodies to demonstrate compliance against regulations; however, the sheer number of regulations means that a fully manual tracing effort can be time-consuming and costly. Unfortunately, the standard Information Retrieval approaches often fail to deliver a relatively complete and accurate set of trace links [14].

An analysis of four different datasets (i.e. CM1-NASA [65], GasCodes [14], AREMA

[14], and HIPAA [1]) showed that in each case, 5 - 13% of the targeted trace links exhibited significant term-mismatches leading to underperformance. A term-mismatch occurs when language in the target document neither matches the language of the source document nor matches project level synonyms defined in a project glossary or standard thesaurus [147]. In fact, this is such a common occurrence, that traceability tools, such as Poirot [29], incorporate features which allow humans to manually add search terms and to filter out unwanted terms. This is illustrated in Figure 2.1, which depicts a HIPAA regulatory code related to access control, traced to CCHIT [24] healthcare system requirements. Requirements are ranked and displayed according to the likelihood of their relevance [147]. In this particular example, the user has added an additional term "authorize", and has filtered out several terms that she felt were causing imprecision in the results. The query was modified accordingly and the trace algorithm rerun to generate an updated list of potentially relevant requirements. These are depicted on the left side of Figure 2.1.

Although this type of modification has been shown to improve the recall and precision of certain trace queries [130], its effectiveness is dependent upon the domain knowledge of the person performing the trace, and also upon their willingness to manually modify the query. Unfortunately, prior studies have demonstrated that users lose confidence in a traceability tool that returns inconsistent or imprecise results, and that human errors are introduced when human analysts are asked to evaluate a long list of candidate links [64, 34]. The approaches described in this chapter therefore represent possible solutions for automating the query-modification process either explicitly prior to running the query or intrinsically as the query is being processed.

2.1.1 Tracing Solutions

We propose a novel approach for integrating ontology into the tracing process for addressing the term-mismatch problem. This **ontology-based** tracing solution captures do-

[1]The technical safeguards of the USAs Health Insurance Portability and Accountability Act of 1996

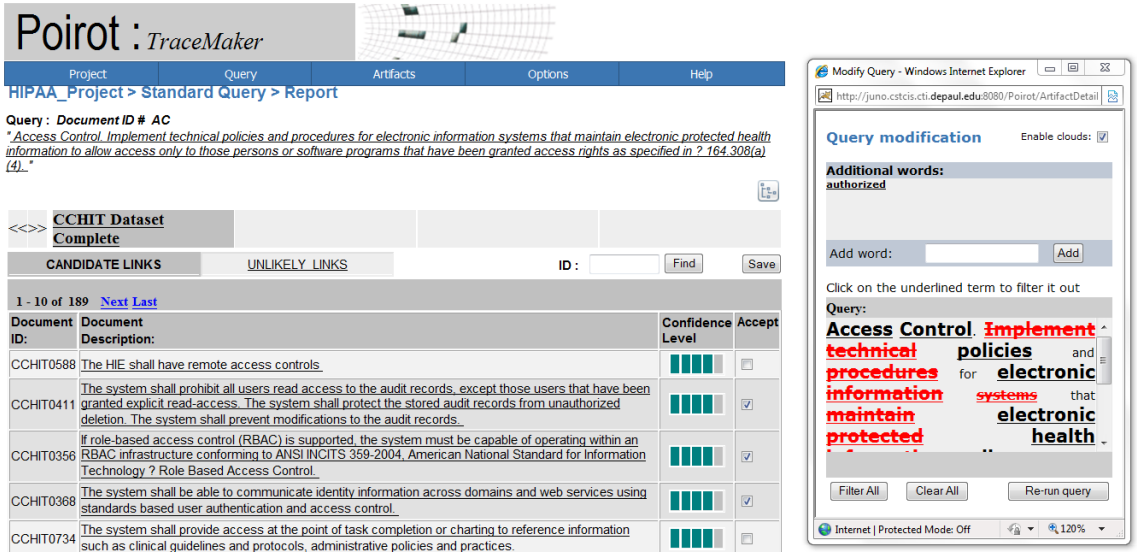


Figure 2.1. HIPAA access control regulation query manually modified in the Poirot tracing tool

main terminology and concepts [54] in the form of an ontology and then uses this knowledge to dynamically augment the text in the query during the tracing process. The trace links are then established between the augmented query and document based on the Vector Space Model (VSM) method [124]. While researchers have previously proposed the use of ontology for such purposes [80], to the best of our knowledge the work we present here represents the first concrete study that demonstrates its efficacy for improving trace accuracy. The examples provided in this chapter utilize an ontology that was constructed manually with semi-automated support [56]. We evaluate the ontology-based tracing methods against the basic VSM and two existing solutions that also attempt to modify or augment the trace query in some way. The other two baselines are the classification technique and the web-mining technique.

The **classification** baseline technique uses a pre-existing, human-validated trace matrix to train a classifier to trace specific regulatory codes. The classifier described in this chapter is an extension of the technique published in a paper entitled "A Machine Learning Approach for Tracing Regulatory Codes to Product Specific Requirements" [32]. Trace-

by-Classification works particularly well in cases where a sufficiently large training set of examples is available, and when the regulations trace to a relatively homogeneous set of requirements.

The **web-mining** baseline technique utilizes information retrieval to mine terms from online domain specific documents in order to replace or augment the original trace query. This technique was initially presented in a paper entitled "A machine learning approach for tracing regulatory codes to product specific requirements" [32] and described and evaluated in depth in a subsequent paper entitled "Towards Mining Replacement Queries for Hard-to-Retrieve Traces" [47]. In general, this approach does not perform as well as the classifier, however, it has the advantage of not requiring a training set of linked requirements and can therefore be used when insufficient training data is available.

The experimental results show that the ontology solution makes noticeable improvements, but requires non-trivial effort to create a suitable ontology for a project. The classification approach returns the best results when sufficient training data is available. Finally, the web-mining technique shows improvements in only a subset of queries. As discussed throughout the remainder of the chapter the three query augmentation techniques offer tradeoffs in terms of performance, cost and effort, and usage viability within each project context.

2.2 Background

2.2.1 Data Sets

The tracing techniques are evaluated against the technical safeguards of the USA's Health Insurance Portability and Accountability Act (HIPAA) of 1996, with a focus on security safeguards. These safeguards, shown in Figure 2.2, describe mandatory features such as access control, audit controls, authentication, and transmission security. The HIPAA security safeguards are traced against the software requirements specifications

(SRS) of ten electronic healthcare systems. During the summer of 2010 we hired an MS Information Systems student with prior industry experience to retrieve datasets of health-care related requirements. He identified a set of 10 systems in the form of open source products, IT healthcare standards, requirements exemplars, and feature descriptions for commercial products. Information for each of these projects, and their relevant source documents are summarized in Table 2.1. In cases where the requirements were extracted from product-level documentation or from online forums the original language and text was retained in order to ensure that no additional terminology was added. Table 2.1 also shows the number of requirements that were considered relevant for each of the HIPAA regulations. Two members of the team (including the original data retriever) separately reviewed all of the requirements, identified ones that were relevant to each HIPAA regulation, and constructed a trace matrix accordingly. Any discrepancies were resolved through discussion between four members of our research team at the time. The retrieval of the requirements, and subsequent construction of trace matrices, was therefore performed internally by authors of the original paper on trace link classification [32]; however, the datasets have since been utilized and validated by an external researcher from CalPoly.

The process of finding health-care related requirements specifications, preparing the datasets in a standard format, and constructing links took a total of 125 hours. For illustrative purposes, the traceability matrix for the Automatic Logoff (AL) regulation is depicted in Table 2.2. It shows that a total of ten AL requirements were found across seven of the ten projects.

In this chapter, we trace HIPPA regulatory codes against Healthcare requirements. Therefore, HIPPA regulations are used as queries and requirements as target documents. We refer to a requirement that is related to regulation q as a *q type requirement*.

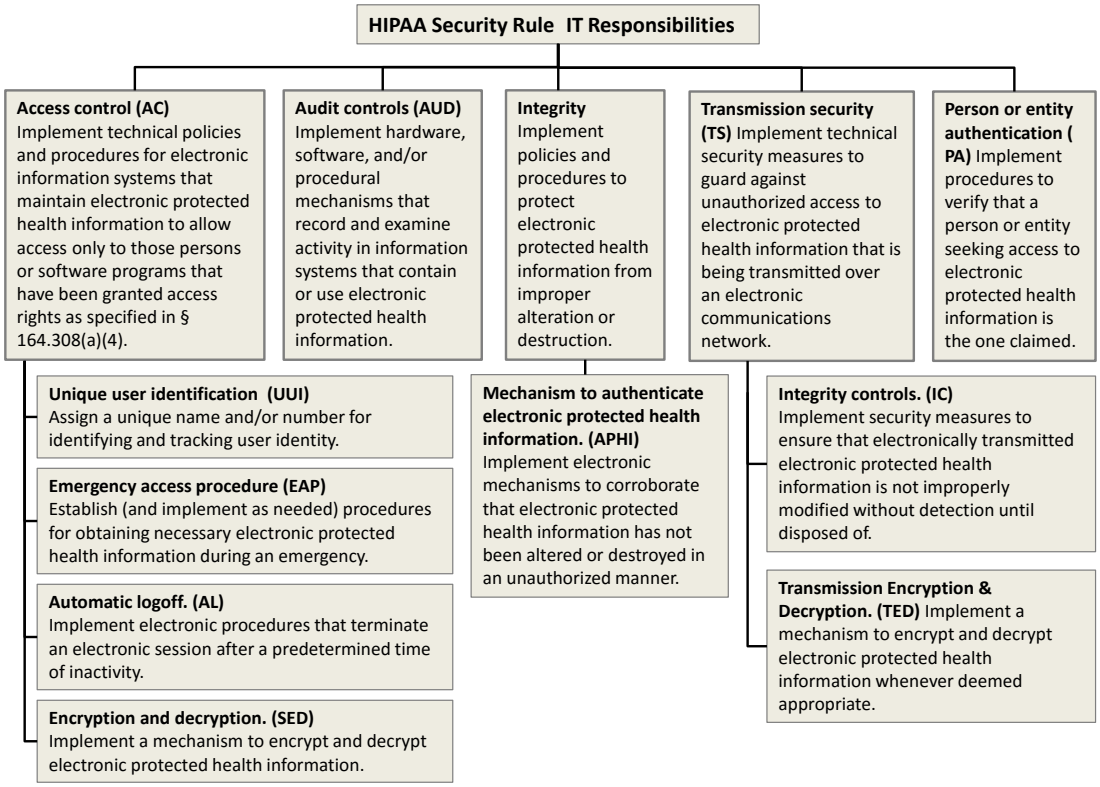


Figure 2.2. HIPAA Security Rules and Responsibilities

TABLE 2.1

HEALTHCARE RELATED DATASETS USED IN EXPERIMENTS

Description	All	AC	AUD	AL	EAP	PA	SED	TED	TS	IC	UUI
Care2x: An open source Hospital Info. System (HIS). Care2x User Manual and Forum. http://www.care2x.org	44	1	1	1	0	1	1	1	0	0	0
CCHIT: Requirements for the Certification Commission for Healthcare Technology http://www.cchit.org	1064	17	33	1	1	12	2	2	2	5	3
ClearHealth: Open source HER. ClearHealth Web Site http://www.clear-health.com	44	1	4	1	0	0	1	1	0	2	1
Physician: Electronic exchange of info between clinicians. Use Cases. hmss.org/content/files/ CTC_use_Case.pdf	147	7	2	0	2	0	0	0	1	3	0
iTrust: Open source HER. Use Cases. http://agile. csc.ncsu.edu/itrust/wiki/doku.php	184	3	35	1	0	6	0	0	0	0	2
Trial Implementations: National Coord. for Health Info. technology (HIT). Use Cases. http://healthit.hhs.gov	100	6	0	0	0	13	0	0	2	4	2
PatientOS: Open source healthcare info. System. PatientOS Web Site http://www.patientos.org	90	1	2	3	1	0	3	1	1	0	1
PracticeOne: A Suite of healthcare info. Systems. Practice One Website http://www.practiceone.com	34	3	1	0	0	1	0	0	1	1	0
Lauesen: Sample EHR requirements. (link no longer available)	66	11	0	1	0	5	0	0	0	3	1
WorldVistA: Veteran Administrations EHR VisA Manuals, EHR Application Features http:/worldvista.org	116	6	2	2	0	4	0	0	0	0	1
Total Counts	1889	54	86	10	4	42	7	5	7	18	11

Column **All** presents the total number of requirements (Reqs) in each dataset. Column **AC** to **UUI** present the total number of HIPAA related requirements (Rel) in each dataset. The last row presents the counts per HIPAA Regulation of all datasets

TABLE 2.2

REQUIREMENTS ASSOCIATED WITH THE HIPAA REGULATION FOR AUTOMATED LOGOFF

Dataset	**Requirement**
ClearHealth	System has a timer allowing automatic logoff
Care2x	System will implement session time outs and use cookies to terminate an electronic session
Lauesen	The system must time out if the user has been away from the system for some time.
iTrust	Electronic session must terminate after a pre-determined period of inactivity. Administrator must be able to specify this period
WorldVistA	The system shall timeout after a period of inactivity.
WorldVistA	The system shall ask the user if the user wants to continue using the system before timing out.
PatientOS	Automatic timeout can be specified by location.
PatientOS	Automatic timeout can be specified by role.
PatientOS	When the system timeouts, the user is returned to the login page to sign in again.
CCHIT	The system upon detection of inactivity of an interactive session shall prevent further viewing and access to the system by that session by terminating the session, or by initiating a session lock that remains in effect until the user reestablish-es access using appropriate identification and authentication procedures. The inactivity timeout shall be configurable.

2.2.2 Basic Vector Space Model Method

Vector Space Model (VSM) is a term-match based information retrieval method first proposed by Salton et al. [124, 125]. It is commonly used in many software tracing problems [89]. VSM represents each *query q* (i.e. the source artifact) as a vector in multidimen-

sional space $q = (w_{1,q}, w_{2,q}, ...w_{N,q})$ in which each term is represented by its own dimension and weighted according to its importance. Documents (i.e. target artifacts) are similarly represented $d_j = (w_{1,j}, w_{2,j}, ...w_{N,j})$. The similarity between query q and document d_j can be computed as the cosine of the angle between the two vectors as follows:

$$sim(d_j, q) = cos\theta = \frac{d_j \cdot q}{\| d_j \| \| q \|} = \frac{\sum_{i=1}^{N} w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2} \sqrt{\sum_{i=1}^{N} w_{i,q}^2}} \tag{2.1}$$

Term t in document d is typically weighted using the *term frequency, inverse document frequency* (tf-idf) computed as follows:

$$w_{t,d} = tf_{t,d} \cdot log \frac{|D|}{|\{d' \in D | t \in d'\}|} \tag{2.2}$$

where $tf_{t,d}$ represents the term frequency of term t in document d, $|D|$ is the total number of documents in the set, and $|\{d' \in D | t \in d'\}|$ represents the number of documents that contain term t.

In preparation for use, each requirement and each regulatory code is preprocessed to remove *stop words*, i.e. commonly occurring words such as “system” and “shall”. Next, because words with similar semantic interpretations often have morphological variants, the remaining words are stemmed to a common root form using the Porter’s stemming algorithm [116].

As introduced in the beginning of this chapter, VSM is only effective when terms co-occur across both the query (regulations) and documents (requirements). But the term-mismatch problem that frequently exists between software artifacts hinders the effectiveness of trace link generation when VSM is used.

2.2.3 Trace Retrieval Metrics

Information retrieval results are typically evaluated using *recall* and *precision* metrics [123], where recall measures the fraction of relevant links that are retrieved and precision measures the fraction of retrieved links that are relevant. For experimental purposes, a threshold score is established such that all links above or at the threshold are retrieved and all links below the threshold are rejected. In order to compare recall and precision results across experiments the F-Measure, which computes the harmonic mean of recall and precision is often adopted. We use a variant of the F-Measure, known as the F2-Measure, which weights recall values more highly than precision. This measure is commonly used to evaluate experiments in the traceability domain where recall is valued more highly than precision. The F2-Measure is computed as follows:

$$F2\ Measure = \frac{5 \times Precision \times Recall}{4 \times Precision + Recall} \tag{2.3}$$

Finally, we adopt another well-used metric of Mean Average Precision (MAP) which evaluates the extent to which relevant links are listed near the top of a ranked list as opposed to lower down the list. First, Average precision (AP) is computed as:

$$AveragePrecision = \frac{\sum_{rank=1}^{|Retrieved|}(Precision(rank) \times relevant(rank))}{|RelevantLinks|} \tag{2.4}$$

where $|Retrieved|$ represents the number of retrieved links, *rank* is the position of the requirement in the ordered set of candidate traceability links, *relevant*() is a binary function assigned 1 if the rank is relevant and 0 otherwise, and $Precision(rank)$ is the precision computed after truncating the list immediately below that ranked position. Mean Average Precision (MAP) is then computed as the mean AP across a set of queries. MAP is a very meaningful metric in the traceability domain as it has the ability to capture a "perfect" trace in which all correct links are listed at the very top. These metrics are used throughout the remainder of this chapter to document and compare results from each of the experiments.

2.3 Ontology-Supported Traceability (OST)

In this section, we propose a novel tracing technique to overcome the drawback of the basic VSM method. It utilizes a domain ontology to modify the trace query, and then adopts the VSM method to compare the similarity between the modified query and the document. The ontology effectively creates a bridge between terms in the query and document that are conceptually related in the context of the project domain. We also discuss the impact of different ontology sources and the impact of variant parameter settings on the performance of the ontology-based tracing method.

2.3.1 Query Modification with Ontology

Query modification techniques aim to make effective connections between a query and a document by adding or replacing terms in the original trace query that also appear in the document. For example, a new term such as *transaction* can be added to the Audit Control (AUD) query based on existing trace links, thereby creating an effective bridge from the AUD regulation to the requirement stating that "System will support transaction logs that will include the MID of the editor, transaction type and transaction date." This requirement would be otherwise missed by the basic VSM method.

A popular way to achieve query modification in general Q&A applications is through building and leveraging a domain ontology. An ontology provides formal definitions of concepts, identifies synonyms, expands acronyms, and captures relationships such as *is-a* and *part-of*. A *domain ontology*, as its name implies, captures the specific terminology and meanings that are common to a specific domain [55]. For example, consider the requirement in Table 2.2, which states that "Automatic timeout can be specified by location." A domain ontology could capture the relationship between *timeout* and *predetermined time of activity*, thereby allowing a trace link to be created between this requirement and the HIPAA Automatic Logoff (AL) requirement. Similarly, in our previous AUD example, the

ontology could capture the concept that an *Audit log* records *event transactions*.

The benefits of using ontologies for generating trace links have previously been proposed, but little empirical work has been performed to evaluate whether it is actually effective. For example, Hayashi et al. [63] and Assawamekin et al. [5] proposed using ontology to connect concepts in source and target artifacts through relationships such as *is-a*, *composed-of*, or *is equivalent to*, allowing trace links to be established based upon general and specific instances of concepts occurring across pairs of artifacts. Instead of permanently replacing the original query with a new one, a domain ontology is used to dynamically augment words and phrases in the original query at query execution time.

There are two primary challenges related to ontology use in traceability. First, there are few appropriate ontologies available. Most existing ones are not readily applicable to traceability tasks because they tend to represent everyday objects and do not include the technical engineering concepts needed to trace requirements and other kinds of entities in software-intensive systems. Second, the domain information within the ontology must be leveraged to identify related concepts so that correct and accurate trace links can be constructed.

2.3.2 Ontology Construction

Ontologies used for traceability purposes must be custom built to support very specific domains of use, a task which is difficult to fully automate and costly to perform. For example, in the HIPAA domain, there is a long-term community effort underway to build a technical ontology to support data queries [52, 53]. The partially constructed ontology defines entities such as *Subject* is a superclass of *Patient* and *Investigator*, *Consent*, *Operation*, and *HealtCareActivity*. Entities are connected through a fairly extensive set of relations that include *labRecord*, *canPerformOperation*, *hasCreator*, and *shareWith*. For example, a *consent hasSubject Patient*. Further it *hasPolicyResults* of type *ConsentRule* which in turn include subclasses of *Permission* and *Obligation*.

Consider the example depicted in Figure 2.3. In both requirements R1 and R2 a trace link should be created to the HIPAA automatic logoff regulation. In the case of R1, terms in the requirement such as "terminate" and "inactive" also occur in the HIPAA regulation, and a trace link can be established using a basic approach such as VSM. In the second case, there are insufficient shared terms; however, if an ontology contains the facts depicted for "time", "terminate", "computer", and "inactive" (among others), then the requirement can be matched to the regulatory code despite only sharing one non-trivial term (session). *Work Station* is associated with *Computer* via the is-a relationship in the ontology, *idle* is associated with *inactive*, and *minutes* is replaced with *time*. While numerous other associations are possible, these are the ones which bridge the gap between the HIPAA regulation and the requirement. The trace link is correctly established because of these concepts in the ontology.

In an ideal world, pre-constructed ontologies for a diverse set of technical domains would be readily available to support Software Engineering activities such as traceability. However, this is not the reality, and furthermore, building a sufficiently complete ontology

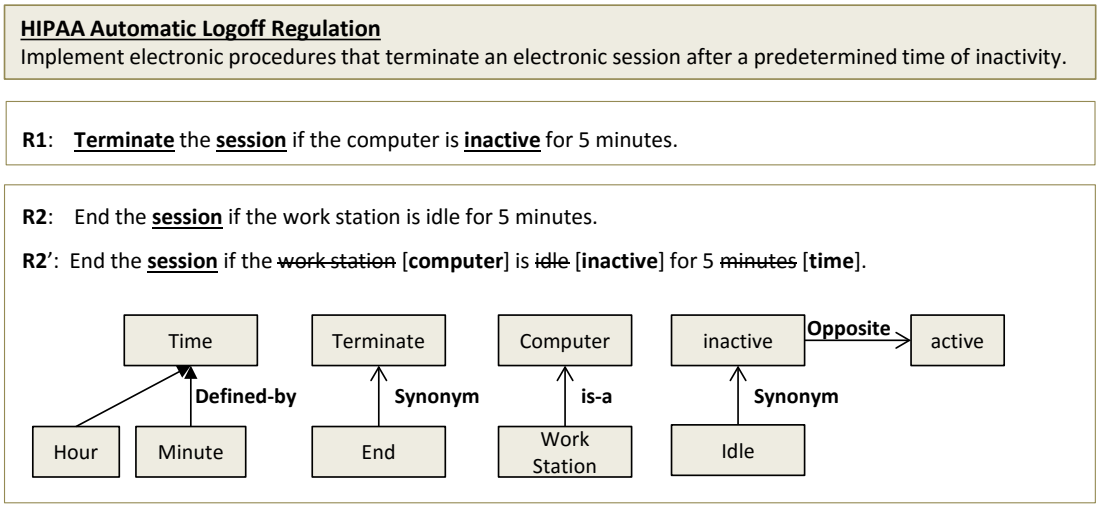


Figure 2.3. Example showing how a small ontology bridges the gap between a regulation and a requirement

for a specific domain is likely to produce conceptual facts which may be superfluous for tracing purposes. For purposes of the study reported in this chapter we therefore manually created the ontology needed to support traceability in a *core project* and then used the resulting ontology for tracing purposes in the remaining projects. We recruited three DePaul graduate students to perform the task of reviewing the HIPAA regulations and their trace links in the *core project* and identifying domain concepts that explain the link.

We provided a semi-automated tool which was designed to assist in this task. First, we used the Stanford Parser [78] to extract nouns, verbs, and noun phrases from each HIPAA regulation and its linked requirement. We refer to these as the artifact's **Key Contents** (KCs). Next, for each pair of source and target artifacts related by a validated trace link from the core project, we generated Candidate Facts (CFs) represented by pairs of KC_{Source} and KC_{Target} elements. For example, given a HIPAA regulation containing the phrases *audit log* and *record*, and a linked requirement containing terms *transaction* and *update*, we used our tool to explore the four candidate facts of *audit log - transaction*, *audit log - update*, *record-transaction*, and *record-update*. Finally, we applied a standard Association Rule Mining (ARM) algorithm [2] to compute the likelihood of each candidate fact, and then used this score to rank the candidate facts.

ARM was applied as follows. Given a set of connected artifact pairs T, a set of words and noun phrases, i.e., items $I = \{I_1, I_2, \cdots, I_k\}$, and an item set $is \subseteq I$, let $T_{fi} \subseteq T$ be the set of artifact pairs that have all the items in is. The *support* of the item set is is defined as $\sigma(is) = |T_{is}|/|T|$. Item sets that satisfy a predefined support threshold are referred to as *frequent item sets*. An association rule r is expressed in the form $X \implies Y(\sigma_r, \alpha_r)$, where $X \subseteq T$ and $Y \subseteq T$ are item sets, σ_r is the support of the item set $X \cup Y$, and α_r is the *confidence* for the rule r given by $\sigma(X \cup Y)/\sigma(X)$. The *lift* measure is defined as $\sigma(X \cup Y)/(\sigma(X) \times \sigma(Y))$. Candidate facts were presented to the user ordered by the *lift* measure.

The user could accept or reject the candidate facts. For all accepted candidate facts

the user specifies relations as either *hierarchical*, *compositional*, and *equivalence* or could create a new type of relation. The user also assigned a *confidence rating* to each accepted fact. This is helpful because certain facts are likely to always hold, whereas others are only true sometimes. For example, the domain fact that *access control is-parent-of role-based access control* is generally true and could be ascribed a confidence rating of 1. On the other hand, the domain fact *activity has-feature event type* is a less general relationship and is assigned a lower confidence score (in this case 0.5). In our current ontology, confidence ratings are determined by the user, utilizing three possible scores of 1, 0.8, or 0.5. As reported in the following section, our experimental results showed these subjective ratings to be quite effective.

Domain ontology were saved in Protégé [135], which is a free, open-source ontology editor developed by Stanford University School of Medicine. The KC_{Source} and KC_{Target} for each fact are represented as entity names connected through the hierarchy of ontology classes or properties of those classes. Part of the domain ontology created from trace links between HIPAA and CCHIT (core project) is shown in Figure 2.4. It is important to note that even with semi-automated support for ontology creation, the process is still currently human intensive and influenced by individual user decisions.

2.3.3 Leveraging Ontology to Generate Trace Links

Once ontology has been built for a domain, it can be used during the trace link generation process to expand terms in the original queries. Our ontology-enhanced technique is built over the standard VSM model. Two KCs in the ontology are considered to be semantically related if they are connected either directly or indirectly. The *Semantic Relatedness (SR)* of two entities can be measured in several different ways. In this chapter we consider two techniques which we define as *Information Content (IC)* based and *Directly Connected Domain Facts (DCF)*.

Information Content (IC) based measures were proposed by Lin [86]. They treat the

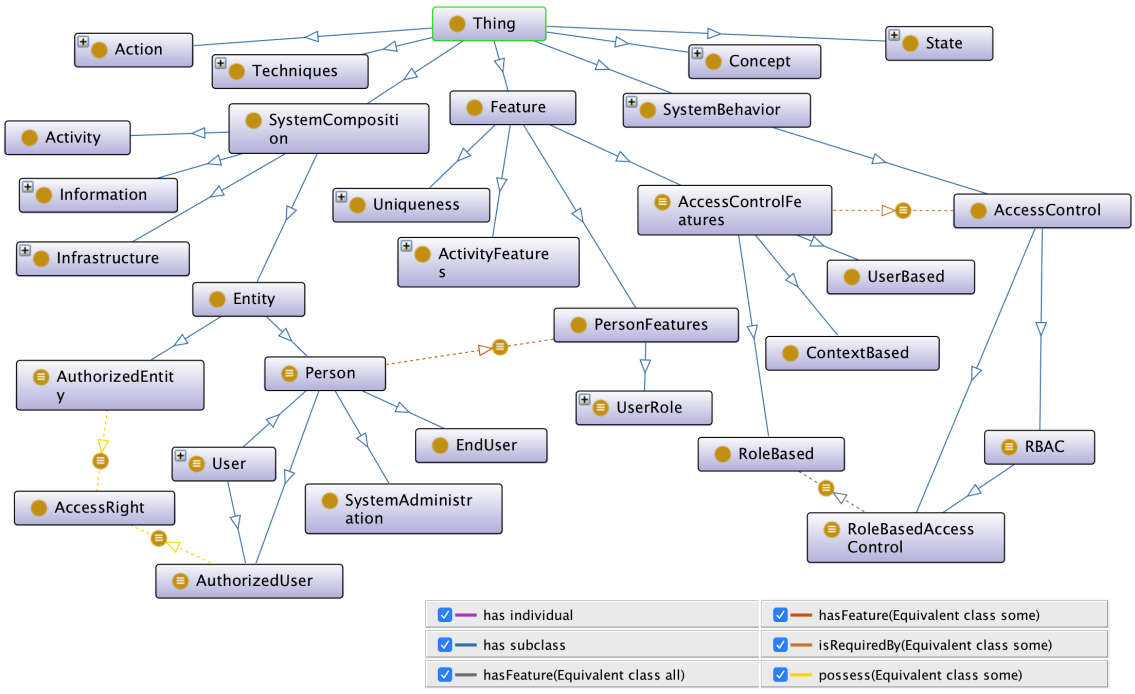


Figure 2.4. Part of the ontology generated from trace links between HIPAA and CCHIT

ontology as a taxonomy and compute SR as follows:

$$SemanticRelatedness = 2 \times logP(E_0)/(logP(E_1) + logP(E_2)) \tag{2.5}$$

where E_0 is the Lowest Common Ancestor of E_1 and E_2 and $P(E_x)$ is the probability that a randomly selected entity belongs to E_x. **Directly Connected Domain Facts (DCF)** consider only directly-related entities and weight their associations according to the user-generated confidence scores.

To process a query, we first check to see if it contains any domain entity names (e.g. "RoleBased" or "Infrastructure" as depicted in Figure 2.4). If entity names exist in the query they are augmented with their associated ontology concepts. Furthermore, a weighting of $SR \times w_{SA}$ is applied, where w_{SA} is an additional weight for controlling the general

influence of the concepts associated through the ontology. For experimental purposes we evaluated each of the semantic-relatedness algorithms at weights of 0.5 and 1.0. Following this step, both the query and documented are represented in the vector space alongside additional dimensions corresponding to their associated domain entities. During the trace retrieval phase, the similarity between the query and the document is then calculated as the cosine of the angle between query and document vectors weighted by the standard *tf-idf* scheme using Formula 2.1 and 2.2. Examples of domain facts used during the tracing process are reported in Table 2.3.

2.3.4 Comparison of Two Ontology-Based Techniques and VSM

We evaluated the efficacy of the IC and DCF techniques on the overall quality of generated trace links. For experimental purposes, we constructed an ontology using the CCHIT dataset as the *core project* as this is the largest dataset and therefore most likely to generate a broad set of generally useful domain facts. The generated ontology was then used to support the generation of trace links between the HIPAA technical safeguards and each of the remaining datasets described in Table 2.1. Each of the Ontology-Based Techniques were evaluated and results are reported in terms of Mean Average Precision in Figure 2.5. Results indicate that the DCF approach for computing Semantic Relatedness outperformed the IC approach at both weightings, with its top performance observed when weighting was set to 1. For the winning method (i.e. DCF-Weighting 1.0) we report Recall, Precision, F2-Measure, and MAP scores against individual HIPAA technical safeguards in Table 2.4.

TABLE 2.3

USING ONTOLOGY TO MATCH REGULATION AND REQUIREMENT AT RUNTIME

Reg	**Sample REQ Traced to Reg**	**Used Domain Facts**
AC	The credentials system employs a standards compliant **RBAC** model so each **user** has **permissions** appropriate to their role.	Access Control is *parent of* RBAC; Person is *equivalent to* user; Access rights is *child of* permission.
AL	Automatic **timeout** can be specified by role.	Automatic logoff *has feature of* timeout.
AUD	The system **audit trail** shall automatically capture system data changes, edits, charged, credits by all **users**.	Audit *requires* audit trail; Activity *is performed by* user.
EAP	When access to a chart is restricted and the **break the glass** has occurred, the system shall provide the ability to audit this override.	Emergency *is a result of the action* break the glass.
IC	The system shall provide the ability to retain data until otherwise purged, deleted, archived or otherwise deliberately **removed**.	Dispose *is equivalent to* remove.
PA	System will generate a secret key - patient's initial **password**.	Authentication *is parent of* password.
SED	The system, when storing PHI on any device intended to be portable/removable, shall support use of a standards based encrypted format using triple-DES, or the Advanced **Encryption Standard**, or their successors.	Encryption *requires* encryption standard.
TED	Everything is transmitted over SSL which offer encryption but also **hashes** to prevent modification in transit.	Encrypt *has technique* hash
TS	The system shall ensure secure **electronic messaging** with patients.	Electronic communication network *is required by* electronic messaging.
UUI	System will generate a **unique** Medical User Identification number.	Identify *requires* uniqueness; Track *requires* uniqueness.

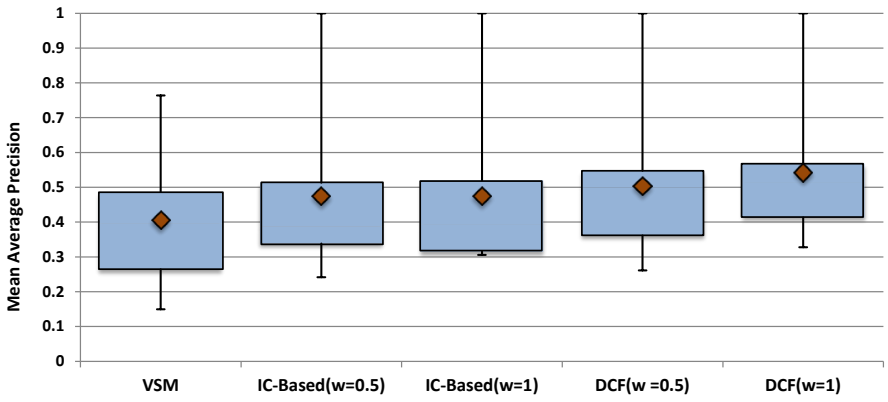


Figure 2.5. A Comparison of VSM versus two Ontology-Supported techniques with different weighting

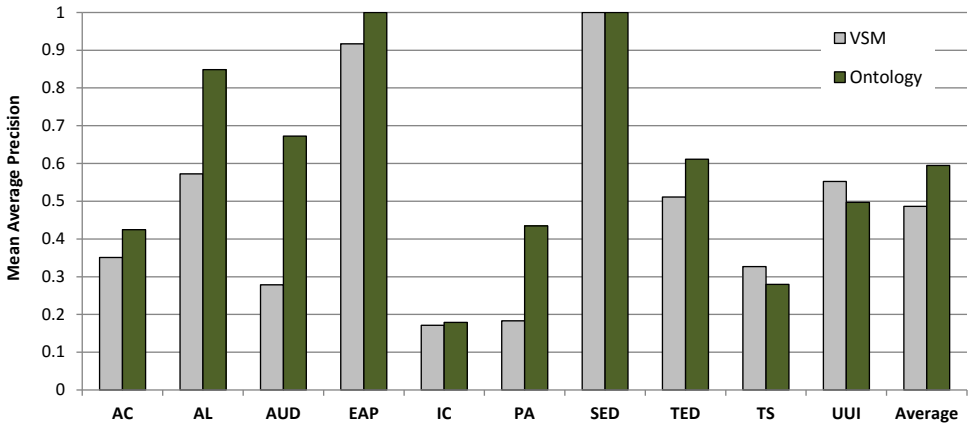


Figure 2.6. A comparison of Ontology-Supported (using DCF-Weighting 1.0) versus the basic VSM approach.

Note: the results shown here are for all projects except CCHIT. As CCHIT was used to build the ontology it is excluded from both Ontology and VSM results.

TABLE 2.4

RESULTS FROM ONTOLOGY-SUPPORTED TRACEABILITY METHOD USING THE WINNING METHOD OF DIRECT DOMAIN FACTS

HIPAA ID	**Recall**	**Precision**	**F2**	**MAP**
Access Control (AC)	0.826	0.394	0.571	0.430
Automatic Logoff (AL)	0.929	0.911	0.901	0.870
Audit Trail (AUD)	0.932	0.648	0.774	0.666
Emergency Access Procedure (EAP)	1.000	1.000	1.000	1.000
Integrity Controls (IC)	0.697	0.147	0.350	0.151
Personal Authentication (PA)	0.966	0.328	0.577	0.393
Storage Encryption & Decryption (SED)	1.000	0.917	0.977	0.896
Transmission Encryption & Decryption (TED)	1.000	0.583	0.845	0.646
Transmission Security (TS)	1.000	0.224	0.537	0.357
Unique User ID (UUI)	0.929	0.515	0.667	0.473

In Figure 2.6 we compare the use of Ontology against the basic VSM method. As CCHIT was used to build the ontology, the ontology results are computed against the nine remaining projects only. For comparison purposes we therefore also report VSM results for only nine projects. Overall, we see that for seven out of the ten HIPAA regulations the use of the CHHIT-constructed ontology delivered improvements over VSM when applied to the remaining nine projects. At the level of the individual query, the ontology improved MAP scores for 36 queries, degraded it for 8 queries, and tied in 18 queries. We report results from our statistical analysis for the Ontology method in Section 2.5.

2.3.5 Impact of Ontology Source

The previous experiment used CCHIT, as the core project; however, to better understand the extent to which ontology built on one dataset could be applied across others from the same domain we conducted a second experiment, in which we systematically used each of the datasets as the core-project from which to construct an ontology. The resulting ontology was then used to generate trace links between HIPAA regulations and requirements in each of the remaining datasets. Results are reported in Table 2.5.

The CCHIT-built ontology outperformed the others in eight of the nine cases, the only exception being the Lauesen dataset which performed best on the WorldVista ontology. CCHIT's winning performance is not surprising as it contains more artifacts and trace links, and therefore increases the chance of documenting domain facts used in other projects. However, while the CCHIT-built ontology outperforms the other ones, it is interesting to note that there were only six cases out of 90 in which use of any ontology delivered worse results than the use of VSM alone. One example is the case of applying the PracticeOne ontology to ClearHealth, where the MAP score dropped from 0.352 for VSM to 0.346.

TABLE 2.5

MAP SCORES ACHIEVED WHEN ONTOLOGY WAS CONSTRUCTED USING A SOURCE DATASET AND APPLIED AGAINST THE TARGET DATASET

Target Dataset	VSM	Source Dataset (Used for Constructing Ontology)									
		Care 2x	CCHIT	Clear Health	Phys-ician	iTrust	Trials Impls	Patient OS	Practice One	Lauesen	World VistA
Care2x	0.764	N/A	**1.000**	0.875	0.792	1.000	0.889	0.792	0.889	0.889	0.806
CCHIT	0.331	0.359	N/A	0.369	0.346	0.359	0.348	0.375	0.362	0.373	**0.380**
ClearHealth	0.352	0.511	**0.541**	N/A	0.378	0.370	0.375	0.484	0.346	0.289	0.438
Physician	0.265	0.265	**0.328**	0.271	N/A	0.280	0.320	0.274	0.265	0.288	0.288
iTrust	0.486	0.535	**0.593**	0.484	0.516	N/A	0.551	0.507	0.526	0.523	0.569
TrialImpls	0.411	0.431	**0.515**	0.449	0.425	0.455	N/A	0.443	0.422	0.419	0.433
PatientOS	0.560	0.508	**0.568**	0.544	0.560	0.498	0.565	N/A	0.498	0.541	0.543
PracticeOne	0.150	0.183	**0.411**	0.216	0.150	0.225	0.161	0.145	N/A	0.300	0.150
Lauesen	0.395	0.495	0.515	0.361	0.394	0.421	0.424	0.395	0.426	N/A	**0.523**
WorldVistA	0.260	0.343	**0.413**	0.208	0.243	0.264	0.260	0.310	0.260	0.283	N/A
# Average		0.403	0.542	0.419	0.422	0.43	0.432	0.413	0.443	0.433	0.458
# Domain Facts		6	50	13	10	10	17	8	7	14	10

2.4 Other Baseline Query Modification Methods

In this section, we briefly introduce two other query modification methods from recent literature [32, 47]. We discuss their strengths and limitations for tracing to the HIPAA regulations. We will conduct a thorough comparison against the ontology-based tracing method in the following section 2.5.

2.4.1 Classification Technique

The classification technique is based on Machine Learning methods. Machine learning methods are particularly appealing for tracing regulatory codes, such as HIPAA safeguards, because the upfront effort of training a classifier can be potentially recouped when those same regulations are applied across additional projects. Although there are many different classification algorithms, we adopted one that we had previously developed for classifying non-functional requirements (NFRs) [31, 30]. The same algorithm has been applied to classify source code snippets which implement architectural tactics such as *heart-beat* or *resource pooling* [104]. Our study showed that the NFR Classifier outperformed standard classification techniques including the Näive Bayes classifier, decision tree (J48), feature subset selection (FSS), correlation-based feature subset selection (CFS), and various combinations of the above [103].

2.4.1.1 Training the Classifier

The classification process requires a training set of regulatory codes, software requirements, and their associated trace links. All data is prepared using the same preprocessing techniques applied to VSM.

During the training phase, each term in the requirements specification is weighted according to the degree to which it represents a HIPAA regulation. The weight is calculated using the formula below:

$$W_q(t) = \frac{1}{N_q} \sum_{d_q \in S_q} \frac{freq(d_q, t)}{|d_q|} * \frac{N_q(t)}{N(t)} * \frac{NP_q(t)}{NP_q} \tag{2.6}$$

where S_q is the set of all q type requirements in the training set, i.e. requirements that are related to regulation q; N_q is the cardinality of S_q. The part $\frac{freq(t,d_q)}{|d_q|}$ computes the frequency at which term t occurs in requirement d_q; $\frac{N_q(t)}{N(t)}$ computes the fraction of q type requirements that contain term t; $\frac{NP_q(t)}{NP_q}$ computes the fraction of projects which include requirements related to regulation q, which also contain term t. A weighting $W_q(t)$ is computed for each term t with respect to q, and terms are then ranked by decreasing order according to $W_q(t)$. Terms that ranked on top are called indicator terms. Additional experiments performed for this work showed that utilizing all terms returned better results. Therefore all terms have been used as indicator terms throughout the experiments reported in this chapter.

2.4.1.2 Using the Classifier

Once indicator term weights have been computed, the classifier computes a score $Pr_q(d)$ which represents the probability that a given requirement d is associated with the regulation q. The underlying assumption is that type q requirements are more likely to contain indicator terms for that type.

Let I_q be the set of indicator terms for regulation q identified during the training phase using Formula 2.6. The classification score that a requirement document d belongs to regulation R is then defined as follows:

$$Pr_q(d) = \frac{\sum_{t \in d \cap I_q} W_q(t)}{\sum_{t \in I_q} W_q(t)} \tag{2.7}$$

where the numerator is computed as the sum of the term weights of all indicator terms for regulation q that are contained in d, and the denominator is the sum of the term weights for all type q indicator terms. In this way, the classifier will assign a higher score $Pr_q(d)$ to a requirement document d that contains more strong indicator terms for q.

TABLE 2.6

ACCURACY OF TRACE LINKS GENERATED USING THE CLASSIFIER

HIPAA Regulation	**Recall**	**Precision**	**F2**	**MAP**
Access Control (AC)	0.855	0.49	0.663	0.499
Automatic Logoff (AL)	1.00	0.774	0.877	0.839
Audit Trail (AUD)	0.962	0.641	0.798	0.718
Emergency Access Procedure (EAP)	1	0.666	0.874	0.638
Integrity Controls (IC)	0.561	0.153	0.273	0.112
Personal Authentication (PA)	0.854	0.675	0.798	0.737
Storage Encryption & Decryption (SED)	1.00	0.75	0.916	0.854
Transmission Encryption & Decryption (TED)	1.00	0.916	0.977	0.958
Transmission Security (TS)	0.8	0.523	0.66	0.481
Unique User ID (UUI)	0.928	0.399	0.563	0.388

2.4.1.3 Evaluating the Classifier

Given the fact that we had only 10 distinct datasets, we adopted a ten-fold cross validation experimental design. In each iteration of the experiment, nine projects were used to train the classifier. The trained classifier was then used to classify requirements in the remaining project. The process was repeated until each project had been tested (i.e. classified) one time. Based on initial experimentation [32], we adopted a multi-classification approach in which all requirements that scored higher than a certain threshold value were classified as relevant to that regulation. An individual requirement could therefore be assigned to more than one HIPAA regulation. As with the previous experiment, threshold values were set individually for each query so as to maximize F2-measure values.

We report classifier results for each HIPAA regulation in Table 2.6. Figure 2.7 com-

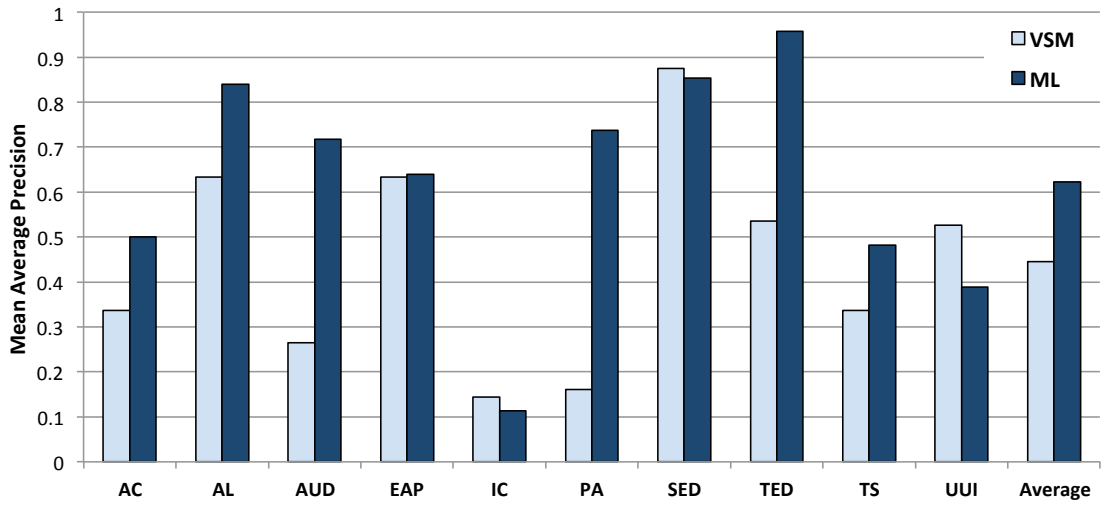


Figure 2.7. A comparison of VSM versus Classification (ML) approach.

pares results against the VSM method. Notably, the classifier improved results in eight out of ten cases. In six of the ten cases (AC, AL, AUD, PA, TED, TS) the classifier approach improved results, in two cases it was closely equivalent (EAP, SED), and in two cases (IC, UUI) it performed worse than the basic VSM approach. Surprisingly, we found that even in the case of relatively small training sets, the classifier performed quite well. The Wilcoxon-Signed rank test with continuity correction showed a significant improvement when comparing the classifier results to VSM (p-value = 0.0002842). This result improves on our previous observations. We attribute the improved performance to our use of all terms as indicator terms to represent each regulation instead of the top 15 used in the previous work [32].

TABLE 2.7

KEY TERMS IN ORIGINAL VS. MACHINE LEARNED MODIFIED QUERIES

Reg	**Original Query**	**Indicator Terms (CHHIT as Test)**	**Indicator Terms (Trial Impl. as Test)**
AC	access allow control electron grant health implement inform maintain person polici procedure program protect right softwar specifi technic	access role user provid secur assign privileg rights controls allow author permiss enforc right consum	access role user secur allow assign privileg rights controls author permiss restrict right provid hospital roles
AL	automat electron implement inact logoff predetermin procedur session termin time	timeout session specifi automat period termin inactivity out time user timer awai logoff return location electron	timeout inactivity session termin automat period out awai logoff timer specifi user location again timeouts time
AUD	activ audit contain control electron examin hardwar health implement inform mechan procedur protect record software	log record audit perform transact everytim interact events user patient access support actions track chang	log record perform audit transact everytim user patient support creat authent time provid access note
EAP	access electron emerg establish health implement inform necessari need obtain procedur protect	emerg phi emergency situations popul workflow access public individu offici individual creat situat electron custom	situations emerg workflow emergency popul access restricted individual offici glass break appropri individu public health
IC	control detect dispos electron ensur health implement improperli inform integr measur modif	format consist data concurr verac integrity deleted problems ensur warn found rbac model standard source	integrity data consist format deleted validity problems concurr doubt metadata record model compliant emploi valid

TABLE 2.7

CONTINUED

Reg	**Original Query**	**Indicator Terms (CHHIT as Test)**	**Indicator Terms (Trial Impl. as Test)**
PA	access authent claim electron entiti health implement inform person procedur protect seek verifi	password authent author user passwords method identifi users provid reset access allow support identification protect	password authent user used author passwords reset allow support method automatically sensit identifi access protect
SED	decrypt electron encrypt health implement inform mechan protect	encrypt databas encrypted tls individu field sl ssl us requir data	encrypt encrypted databas standards field individu sl tls ssl aes 3des us hashing encryption pda
TED	appropri deem electron encrypt health implement inform mechan protect	ssl sl encrypt hash server support modif prevent option everyth offer transit transmit client us commun	ssl sl encrypt hash open server everyth transit offer modif support client option transmit tls internet
TS	access commun electron guard health implement inform measure network protrect secur techni transmiss	transmission secur secreci sender hitsp maintain content deliveri assum construct electron s privaci transport delivery	secur messag ensur electron browser html delivery deliveri confirm perimet ehrs recipient networks web interfac
UUI	assign ident identif identifi number track uniqu user	uniqu number identifi correctli identif systems medic user usernam requested authentication patients password protocol enforc method	uniqu number identif identifi user hies ldap usernam lightweight ensur us enforc password authentication patients

Note: the unstemmed version of the original query is shown in Figure 2.2

To understand why the classifier performed well in the majority of cases, we examine the original query terms alongside two sets of indicator terms in Table 2.7. The first column lists original query terms, while the next two columns list the top 15 ranked indicator terms learned when CCHIT (column 2) and Trial Implementations (column 3) were set aside for

testing. Column 2 indicator terms would therefore be used to test the CCHIT dataset and Column 3 for Trial Implementation etc. On average each HIPAA regulation has 111 indicator terms, although many are weighted with very low scores. The terms learned from the training set provide a far richer explanation of the HIPAA regulations, than the original query. Using the modified query can therefore result in significant improvements in MAP scores.

We analyze the two cases in which the classifier performed worse than VSM (i.e. IC and UUI). In the case of Integrity Constraints (IC) there are 18 requirements dispersed across six datasets. However, the requirements are very diverse and cover topics such as integrity of patient data, retention and purging of data, and sending acknowledgements once transmitted data is correctly received. In the second case of Unique User ID (UUI), the requirements integrate concepts related to unique IDs with a broad range of additional concepts, again leading to diversity of requirements. This analysis suggests that the Classification technique is most effective when requirements provided in the training set are relatively homogeneous. In both the IC and UUI regulations, the HIPAA guidelines could benefit from further refinement.

Acquiring or developing training sets can be time consuming and is not always feasible; however, our results suggest that even relatively small training sets can be effective. Furthermore, constructing a training set might not be that difficult for an organization that builds multiple products that all comply with similar regulations. Another alluring option is for certifying bodies to provide trained classifiers which organizations could use in order to check their systems for compliance.

2.4.2 Web-based Query Augmentation

The second baseline method augments the terms in a trace query through web-mining approach. The process is illustrated in Figure 2.8. A regulatory code is passed to one or more standard search engines, in order to search for, and retrieve, relevant documents. Any

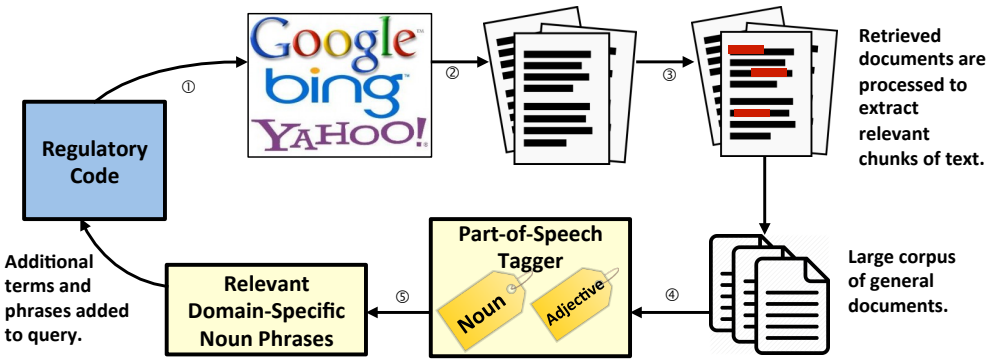


Figure 2.8. Query modification technique

documents which are primarily graphical in nature or composed mainly of advertisements are automatically discarded.

Each of the retrieved documents is partitioned into sections by splitting it into overlapping chunks of length *chunkLength*. A parameter, *chunkOffset* determines the distance in characters between the starting point of the previous chunk and the starting point of the current chunk. Creating overlapping chunks ensures that unfortunate partitioning decisions do not prevent the algorithm from recognizing the most relevant section of text.

For each document, we use VSM to compute the similarity between the chunk and the original HIPAA regulation in order to identify the chunk which exhibits the highest similarity. This most relevant chunk is then analyzed further using an approach we previously developed to extract project glossary terms and phrases from requirements specifications [147]. We first parse the text of the relevant chunk to extract nouns and noun phrases useing QTag, a part-of-speech (POS) tagger [141]. A typical trace query produces between 5,000 and 10,000 nouns and noun phrases. Examples taken from HIPAA's audit control regulation are depicted in Table 2.8.

We then compare the frequency with which nouns and noun phrases appear in requirements related to each HIPAA regulation versus their occurrence in a general domain corpus. For purposes of these experiments, we constructed a general domain corpus from a

TABLE 2.8

SAMPLE TERMS GENERATED FROM THE HIPAA AUDIT CONTROL REGULATION

Term	**DS**	**DTF**	**Term**	**DS**	**DTF**
audit	67.2	5.1	work	14.7	0.5
control	9.0	5.1	board	63.7	0.5
manag	8.5	2.0	respons	8.2	0.5
secur	28.2	1.7	technolog	20.4	0.5
report	5.5	1.4	access	3.9	0.5
risk	473.0	1.4	record	6.7	0.5
system	16.7	1.3	time	1.2	0.5
process	5.1	1.3	audit control	1000.0	0.5
auditor	88.5	1.2	committe	1000.0	0.5

collection of over 50 e-books and other online documents that covered topics as broad as science fiction, business, nature, self-help, and even romance. We compute *Domain Term Frequency (DTF)* and *Domain Specificity (DS)* metrics to rank and filter terms for use in the augmented trace query. The metrics are defined as follows:

- **Domain Term Frequency (DTF)** computes the normalized term frequency for term t across multiple documents as follows:

$$DTF(t) = \sum_{d \in D} (freq(t, d)/|d|) \tag{2.8}$$

  where $freq(t, d)$ is the total number of occurrences of term t in a given domain-specific document d, $|d|$ is the length of that document expressed as the total number of terms in d, D represents the domain document collection..

- **Domain Specificity (DS)** measures the extent to which a term is specific to documents in the targeted domain in comparison to the frequency of its more general use

[147]. We first compute the domain specificity of term t in document d, $DS(d,t)$, as follows:

$$DS(t,d) = ln[\frac{freq(t,d)}{\sum_{u \in d} freq(u,d)} / \frac{freq(t,G)}{\sum_{v \in G} freq(v,G)} \tag{2.9}$$

where the first element $(freq(t,d))/(\sum_{u \in d} freq(u,d))$ is the normalized number of occurrences of term t in the domain-specific document d, and the second element is the normalized number of occurrences of t in the general corpus of documents. Domain Specificity (DS) is then calculated as the average value of all domain specificities from each document from the collection:

$$DS(t) = \frac{1}{|D|} \sum_{d \in D} DS(t,d) \tag{2.10}$$

In cases where a term is not found in our general corpus, we consider it highly domain specific and assign it the maximum DS value of 1000.0

DS and DTF metrics are computed for each term. Terms exhibiting domain specificity below a predefined threshold are filtered out and the remaining terms are ranked in descending order of term frequency. In the web-mining approach, the original query is entirely replaced with the set of words and phrases that score above the threshold in terms of DS and DTF scores.

2.4.2.1 Applying Web-Augmentation to HIPAA Regulations

A series of experiments, originally reported by Gibiec et al. [47], were conducted to evaluate and fine-tune the query augmentation process and related algorithms. To support these experiments we developed a Java based tool. For each HIPAA regulation it issued a Google, Bing, and Yahoo search. The retrieved documents were parsed, domain-specific phrases were identified, and the top terms exhibiting appropriate levels of term frequency (DTF) and domain specificity (DS) were selected. The tool outputs the set of reconstituted queries, which are passed to VSM in order to generate trace links. Examples are depicted in Table 2.9.

TABLE 2.9

KEY TERMS IN ORIGINAL VS. WEB AUGMENTED QUERIES

Reg	**Original Query**	**Modified Query**
AC	access allow control electron grant health implement inform maintain person polici procedure program protect right softwar specifi technic	access control door reader permiss control system com role lock ident kei
AL	automat electron implement inact logoff predetermin procedur session termin time	logoff post shutdown com auto logon inact forum password login
AUD	activ audit contain control electron examin hardwar health implement inform mechan procedur protect record software	audit risk auditor procedur review complianc govern depart board audit control
EAP	access electron emerg establish health implement inform necessari need obtain procedur protect	emerg procedur health plan implement depart emerg access fire offic build
IC	control detect dispos electron ensur health implement improperli inform integr measur modif	integr
PA	access authent claim electron entiti health implement inform person procedur protect seek verifi	authent entiti health procedur ident implement certif identif password kei
SED	decrypt electron encrypt health implement inform mechan protect	encrypt kei algorithm password disk bit drive certif cipher decrypt
TED	appropri deem electron encrypt health implement inform mechan protect	encrypt decrypt kei algorithm password cipher string byte bit
TS	access commun electron guard health implement inform measure network protrect secur techni transmiss	secur encrypt transmiss kei health risk integr email data com
UUI	assign ident identif identifi number track uniqu user	identif password authent kei login mail card com

Note: the original query shown here is the preprocessed form of the HIPAA regulatory codes shown in Figure 2.2.

We configured the web-mining approach through conducting a series of experiments designed to evaluate the impact of different settings on MAP [2]. As a result, in the experiment reported in this chapter, we choose to use all three search engines, to set the number of documents to retrieve as 10, to customize the chunk-size for individual trace queries, and to set the threshold values of DTF as 0.1/0.2 and DS as 40.

2.4.2.2 Analysis

We report final results for our approach in Figure 2.9. In six of the ten regulations (i.e. AC, AUD, EAP, IC, PA, and SED), Web-Mining improved accuracy of the MAP scores over the basic VSM method. In four remaining cases (i.e. AL, TED, TS, and UUI) it did not. A small increase in average results was observed with VSM returning an average MAP of 0.445, compared to 0.493 for the Web-Mining technique. The Wilcoxon-Signed rank test with continuity correction failed to show a significant improvement when comparing the classifier results to VSM ones (p-value = 0.05955).

An analysis of the results suggests that the web-mining approach works best when regulations describe specific practices or topics which are well understood and are clearly described in publicly retrievable documents. The Unique User Identification (UUI) regulation is an example of a HIPAA regulation which underperformed with the web-mining approach. Its MAP score dropped from 0.526 with VSM to 0.251 with web-mining approach. An analysis of the modified query showed that the web-mining approach retrieved a number of very general terms such as *mail*, *card*, and *com* from the online documents which caused a drop in precision. Integrity Controls (IC) is another interesting case. While the original VSM score was very low at 0.144, the web-mining approach failed to deliver much improvement because the original IC query was reduced to a single word *integ* in the modified query. This indicates that few commonalities were identified across the retrieved documents.

[2]The details of the experiments are reported in the original paper [62].

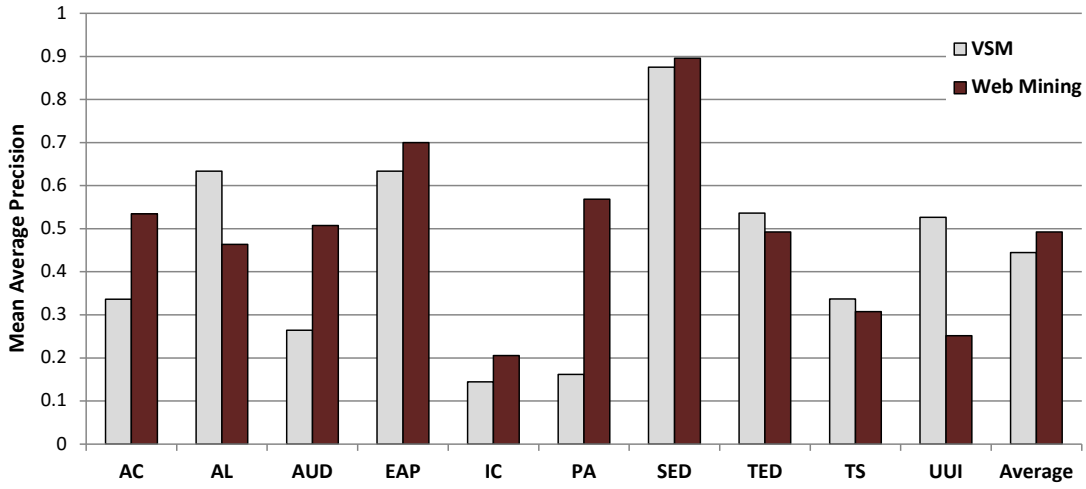


Figure 2.9. VSM versus Web-Mining technique

One of the primary benefits of web-mining is that no upfront training set is required. It can therefore be used without much prior human effort. However, our experiments indicate that it showed only limited, statistically insignificant, improvements. On the other hand it could be a useful option to add into a tracing tool, perhaps as a feature for recommending additional terms to include in a human modified trace query. The user could then accept or decline the recommendations for improving the trace query.

2.5 Comparing Ontology-Based, Machine Learning, and Web-Mining Approaches

In previous sections we have presented three distinct techniques for augmenting queries, namely (1) an ontology-based approach in which domain associated terms were used to expend the original trace queries, (2) a machine learning approach in which a classifier was trained to recognize indicator terms related to specific HIPAA regulations, and finally (3) a web-mining approach in which queries were augmented with knowledge retrieved from mined domain documents. We now make a head-to-head comparison of their effectiveness

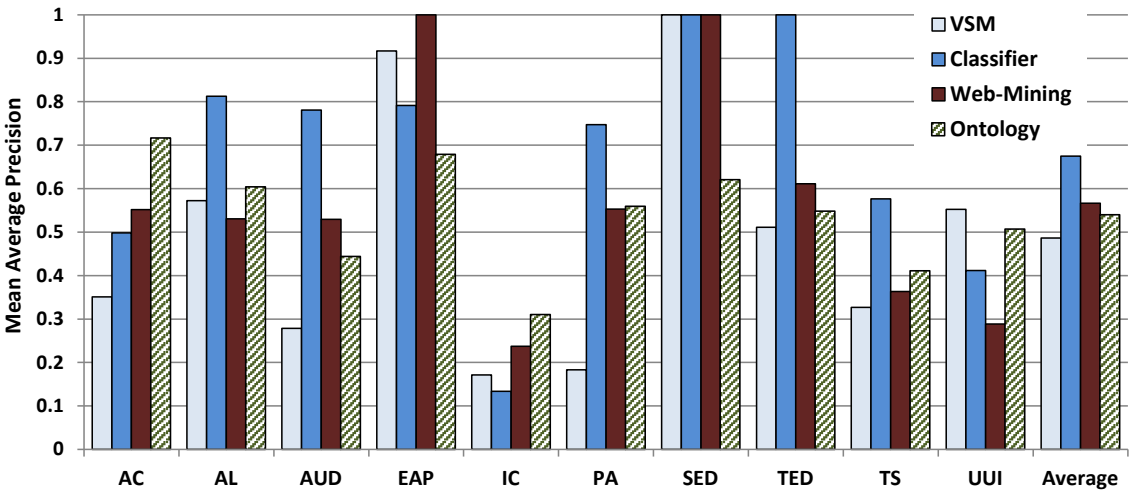


Figure 2.10. Comparison of three techniques by HIPAA type.

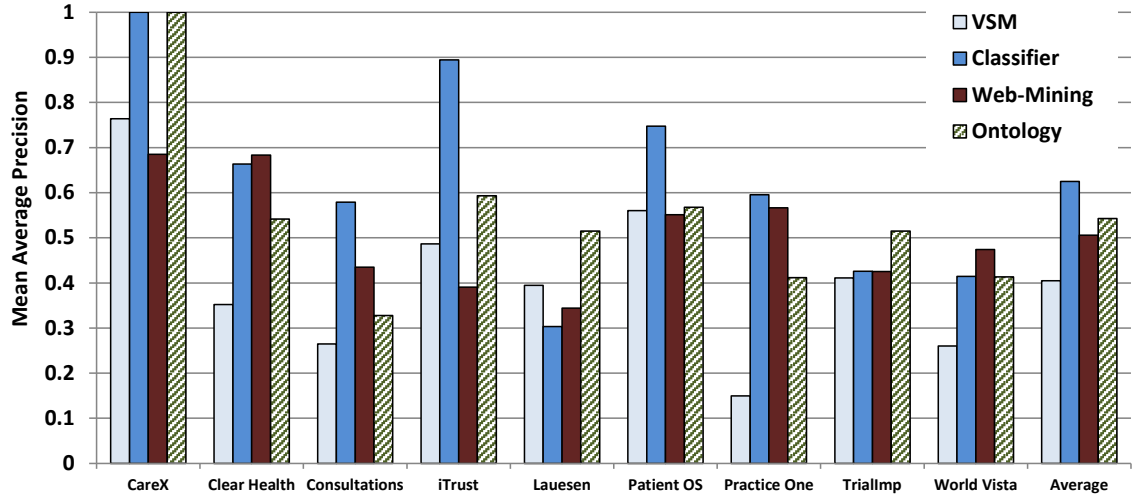


Figure 2.11. Comparison of three techniques by project.

Note: for both Figures 2.10 and 2.11 results are reported for nine projects only. CCHIT results are excluded because CCHIT data was used to build the ontology.

for the HIPAA datasets.

2.5.1 Pairwise Statistical Analysis

Because our ontology-based approach requires one dataset (namely CCHIT) to be set aside, we perform this direct comparison using only nine datasets and exclude CCHIT from reported results for all four techniques. We report MAP results categorized by both HIPAA regulations in Figure 2.10 and projects in Figure 2.11.

Examining the results by project, we can see that the classifier was the overall winner in four projects (Consultations, iTrust, PatientOS, and Practice One), web-mining won in only one project (Clear Health), ontology won in one project (TrialImp), and the classifier tied with ontology in one project (CareX). Examining results by HIPAA type, we observe that the classifier won in five cases (AL, AUD, PA, TED, and TS), web-mining won in one case (EAP), and ontology won in one case (AC). Three techniques of VSM, classifier, and web-mining tied in one case (SED).

Because we wished to compare all four techniques (including the basic VSM) against one another we conducted the following statistical tests. First we performed the Kruskal-Wallis test to determine if statistically significant differences exist between our four techniques. The test achieved chi-squared = 9.1426, df = 3, p-value = 0.02745 which indicates that distinct results were achieved. We then performed a pairwise comparison using Pairwise Wilcoxon test with Bonferroni p-value adjustment. The Bonferroni adjustment reduces the overall likelihood of incorrectly rejecting one or more null hypotheses by chance due to making multiple comparisons. Results, which are depicted in Table 2.10 indicate the only statistical difference is observed between the VSM method and classifier method.

TABLE 2.10

PAIRWISE WILCOXON TEST RESULT

	VSM	Classifier	Ontology-Based
Classifier	0.035	-	-
Ontology-Based	0.272	1.000	-
Web-mining	0.730	0.553	1.000

2.5.2 Discussion of Results

Our reported results clearly show that in the case of the HIPAA datasets presented in this chapter, the classifier performed best followed by the ontology-based approach. However, the HIPAA regulations exhibit very specific characteristics, which will not hold across all regulations. HIPAA regulations are relatively simple, for the most part describe distinct concepts, and are applied across multiple projects making it plausible to construct a training set which can be used by the classifier. Other regulations, such as Sarbanes-Oxley [126], the Payment Card Industry Data Security Standard (PCI DSS) [113], and other types of healthcare related regulations such as ISO/TS 21547:2010, Health informatics security requirements for archiving of electronic health records [73], have quite similar characteristics, and therefore may be likely to perform in a similar way.

It is clear that characteristics of different types of regulations could strongly influence the choice and viability of trace query augmentation solutions. While there are many regulations that are similar in nature to HIPAA ones, there are also others that contain more technical definitions, use “legalese”, and describe quite complex behaviours that include constraints and cross-references [16]. Consider one example of a software-intensive Positive Train Control system operating in Canada which needs to comply with NORAK regulations. One NORAK regulation states that “Gate Arm Clearance Time from a stop equals the time for the design vehicle to accelerate and travel completely through the gate clear-

ance distance" while another states that "The total distance the vehicle must travel to pass completely through the clearance distance ... is computed using the following formula: $s = cd + L$ where: s = distance the road vehicle must travel to pass through the grade crossing clearance distance, (m); cd = grade crossing clearance distance; L = length of the grade crossing design vehicle." Such regulations are clearly different in nature to the HIPAA technical safeguards, and even if training sets were available, the nuanced differences between such regulations would make it relatively difficult to train an effective classifier even though ontology use may still be effective [57, 58].

We document tradeoffs, startup costs, applicability, and constraints for the three query augmentation techniques in Table 2.11.

TABLE 2.11

AN SUMMARY OF THE COSTS, BENEFITS, AND CONSTRAINTS OF APPLYING VARIOUS QUERY AUGMENTATION TECHNIQUES

Technique	**Usage Steps**	**Cost**	**Applicability**	**Constraints**
Ontology	Acquire or construct relevant domain ontology. Instrument the traceability environment to leverage ontology facts.	Ontology construction for 11 HIPAA regulations using the CHITT dataset took 14 person hours (10 hours identifying domain facts, and 4 hours constructing ontology).	Broadly applicable wherever ontology is available.	Domain-wide ontology construction is costly and effort intensive.
Classification	Constructing a training set. Requires a tool (e.g. Weka or TraceLab) for training and utilizing a classifier.	Training set construction took 125 person hours). Costs can be recouped if an organization builds multiple software products in a domain.	Training set size must be sufficiently large, and requirements within the category must be relatively homogenous.	ROI only realized if similar trace queries are applied across multiple projects.
Web Mining	Instrumenting the environment to search for relevant documents, analyze the text, and to identify replacement terms for the original query.	No additional manual effort is required beyond initial instrumentation. No training set required.	Works best when the topic(s) covered by the trace query are clearly defined and explained in publicly available web documents or pages.	Unlikely to be effective for highly complex and/or relatively obscure queries (e.g. for Positive Train Control).

2.6 Threats to Validity

In this section we describe specific threats to validity for the reported study. Such threats fall under the categories of construct validity, internal validity, and external validity [143].

2.6.1 Construct Validity

The goal of this study was to evaluate whether the augmented trace queries were more accurate than those generated using the original HIPAA regulations as the queries. To determine this we first needed to establish an accurate reference set that contained the correct set of trace links from HIPAA regulations to each of the ten software requirements specifications. Such reference sets were not provided to us, and we therefore had to construct them ourselves. The data sets were initially collected and constructed by a graduate Information Systems student and trace links were collaboratively and systematically created and vetted by four members of our research team. Unfortunately creating trace links is not an exact science, and while some trace links are obvious, there are other cases in which judgment calls must be made. These borderline cases were resolved through discussion. We point out that the presence of borderline cases is usually not caused by lack of expertise, as domain experts often disagree over these borderline cases too. To evaluate the quality of the generated trace links, we needed to measure the extent to which trace links matched the reference set. Although there are numerous metrics that can be used, we adopted the Mean Average Precision (MAP) metric described in formula 2.4. This is a well-accepted metric for evaluating trace results, and measures the extent to which the correct links were placed at the top of an ordered set of candidate links [131].

2.6.2 Internal Validity

The results reported from our experiments show that under certain circumstances one query performs better than another one. The primary focus of the work in this chapter was on reporting the methods and results of using three different query enhancement techniques. We applied all three techniques to the same datasets and evaluated results using the same metrics. However, for each individual technique we made configuration decisions which may have influenced the results. In all cases, these decisions were made following rather extensive informal experimentation. We have documented these for each of the three techniques.

2.6.3 External Validity

External validity refers to the extent to which results from the study can be generalized across the entire domain of study. As discussed in the previous section we are not able to claim generalizability for tracing the diverse set of regulatory codes describing elements such as electrical, water, or power regulations, that apply across the systems in engineering domain.

The experiments and techniques described in this chapter focused around tracing HIPAA security safeguards; however it is well known from the data mining literature that results are highly sensitive to subtle nuances of the datasets. The question is therefore to what extent the study of HIPAA regulations is generalizable across a broader set of regulatory codes such as Sarbanes-Oxley [126], the Payment Card Industry Data Security Standard (PCI DSS) [113], and other types of healthcare related regulations such as ISO/TS 21547:2010, Health informatics security requirements for archiving of electronic health records [73]. Due to the significant time and effort needed to construct datasets, traceability reference sets, and to build ontologies, we were only able to conduct experiments with HIPAA regulations. It is likely that different techniques may perform better on different types of regulatory codes. The contribution of this chapter is therefore in presenting

different query augmentation techniques and evaluating them within one specific context. Additional work is needed to evaluation them against a more diverse set of regulations.

On the other hand we claim our approach to be generalizable for tracing HIPAA regulations across a wide range of software requirements specifications. The reported experiments were based on 10 independent requirements specifications taken from entirely different sources. In all cases, the requirements text was extracted directly from source documents, and only very common words were added during the specification process that would either be ignored as stopwords or assigned insignificant weightings by the tf-idf algorithm. Furthermore, we have reported results for each individual project, and have shown that the benefits extend across most of the projects. The results also show that there are cases, i.e. specific projects, that do not trace as well as others, because they do in fact use terminology that is different from the HIPAA regulations and from the terminology used in other projects. This should not be seen as a threat to validity, but rather as a realistic observation and very real limitation of the effectiveness of any trace-retrieval technique.

2.7 Related Work

In this section we summarize key related work in the area of query augmentation – first describing basic techniques, then presenting traceability solutions which leverage external knowledge to improve accuracy, and finally discussing the use of ontology for tracing purposes.

2.7.1 Basic Tracing Techniques

Many different researchers have investigated automated approaches for dynamically generating trace links using standard techniques such as the Vector Space Model (VSM) [64], [65], Latent Semantic Indexing (LSI) [4], [96], [97], and Latent Dirichlet Allocation (LDA). Of these techniques there is no clear "winner" across a diverse collection of datasets [89], [41], [93]. Numerous enhancements have been suggested, but fully auto-

mated solutions rarely outperform the original methods. For example, Sultanov and Hayes proposed the use of reinforcement learning but were not able to achieve more than 55% recall in either of the tested datasets [136]. They also proposed the use of AI swarm technologies, and reported improvements over a VSM baseline when applied to one dataset but decreased accuracy in another [137]. In our own prior work we proposed a variety of clustering techniques to enhance tracing results, but only observed small improvements in trace quality [29].

2.7.2 Augmented Tracing Techniques

Another class of tracing techniques include additional information and/or guidance into the tracing process. These techniques include rule-based approaches [134], scenario and test case-based methods [39], event-based approaches [28], and policy-based methods [106]. However much of this work has focused on generating traces between documentation and code [3, 13] or across artifacts such as requirements, design, code, and test cases within a project. Furthermore, techniques such as scenario or policy- based approaches are more appropriate for tracing within a project than for tracing external documents such as regulatory codes.

While the idea of augmenting search queries through web mining is not new, prior work has focused on user-defined queries for purposes of retrieving information. Such queries typically contain between 2.4 and 2.7 terms [43] and are repetitious across multiple users. In contrast, trace queries often contain 10-100 terms, utilize technical and legal terms, and are used infrequently by a small group of trace users. Despite these differences there is a large foundation of work in the area of query expansion that is relevant to the work we propose in this chapter. For example, Hu et al., used Wikipedia to augment users' web-based queries [71]; however based on our initial analysis of the traceability problem, websites such as Wikipedia, do not currently provide sufficient information to augment a rich and diverse set of domain specific traceability queries. Broder et al. [19] and Shen

et al. [129] have used more general web knowledge to augment queries; however their techniques are primarily designed to expand a short query with additional and potentially disambiguating terms. In contrast, our approach needs to both add and remove terms from an original query.

Researchers in the area of Feature Location have also explored the use of query augmentation techniques. The problem they address is somewhat different from ours, because the target of their search is source code, which is highly structured. Feature location represents a subtly different task from requirements traceability. Its goal is to find source code *related to* a query, whereas the goal of tracing regulations to requirements is that of find requirements which contribute to the *satisfaction* of the regulations. Nevertheless, there are many commonalities. For example Hill et al. focused their feature location searches on noun phrases, verb phrases, and prepositional phrases extracted from method and variable names and organized into a hierarchy [67]. User queries were expanded using the hierarchy. Abebe and Tonella extracted domain facts from the structure of the source code and stored them in a basic ontology [1]. They allowed developers to use the ontology to formulate more precise queries. Finally, Dasgupta et al. used external documentation to augment the corpora of a search space in order to improve its searchability for traceability purposes [36]. They demonstrated an increase in accuracy ranging from 8 to 31%.

Other researchers such as Yurelki et al., proposed collaborative recommender systems which suggest query refinements to the user based on other users' similar queries [145]. Others have augmented queries with different types of knowledge such as folksonomy or taxonomy tags [108]. For example, Gabrilovich et al. classify initial search results according to an extensive taxonomy and then improve search results based on the prior history of queries in the designated categories [43]. Unfortunately, trace queries cannot directly benefit from such techniques as they rely on a large user base and a historical log of related queries.

Hayes et al. [65] and Lucia et al. [91] both evaluated the use of the well-known Rocchio

technique [8] to capture relevance feedback in order to improve trace quality. The Rocchio algorithm, utilizes relevance feedback captured from an initial set of traceability links to modify the underlying representation of the query. We previously introduced the Direct Query Modification (DQM) approach [32, 47, 130], which allows a user to directly modify the trace query by filtering out unwanted terms and adding additional terms and synonyms.

More recently Gervasi and Zowhgi used affinity mining for traceability purposes. Their approach also uses a training set of trace links and uses these to learn affinities between terms in source and target artifacts. Future trace links are generated based on this set of learned affinities [46]. Their approach is very similar to a technique we previously presented which leverages relationships between terms that occurred in linked source and target artifacts to generate future trace links [38]. Gervasi and Zowghi evaluated their work against only one dataset - CM1 which is also used in our dataset. They also used a leave-one-out experimental design and their approach returned precision values of 85% at recall of 90%. Given our analysis of the CM1 dataset and the dearth of association rules we were able to mine, we are unable to explain or duplicate such results.

2.7.3 Ontology in Traceability

There is a small body of work in the area of ontology use for Traceability. Kof et al. extracted domain-specific concepts from the set of traceable artifacts, and then used WordNet to find synonyms in order to map similar concepts and to establish trace links [80]. The major contribution of their work is their approach for extracting an ontology from a requirements specification; however they did not provide a rigorous analysis to evaluate whether the use of the ontology actually improves the quality of the generated trace links. Li at al. also explored the use of an ontology for traceability purposes [85]. Their approach used knowledge from the ontology to establish connections between concepts in source and target artifacts. The strength of the dependency was related to the distance between concepts in the ontology, and the overall similarity score between pairs of artifacts was

increased accordingly. This approach was tested on only one dataset and led to only small improvements in trace accuracy.

Zhang et al. published a paper entitled "Ontological approach for the semantic recovery of trace links between software artefacts" [146]; however ontology is used to represent concepts such as *design patterns*, *sentences*, *paragraphs*, and *classes*, as opposed to domain concepts such as *train* or *signal* found in the domain being traced.

Assawamekin et al. utilized Natural Language Processing techniques to construct separate ontologies from each stakeholder perspective and then to discover a mapping between them [5]. The ontology is expressed in first-order logic and the mapping is solved by a SAT-solver. Their approach is designed to create links between different stakeholder perspectives; however the authors provided only a simple example, and did not empirically evaluate their approach.

Existing tracing tools such as Poirot [87] include capabilities to expand acronyms and utilize lists of matching words in an attempt to bridge the semantic gap. In most cases, these synonyms and acronyms are extracted directly from project documents. Such approaches are not designed to capture the rich set of relationships between concepts that can be modeled through the use of ontology.

2.8 Conclusion

The work described in this chapter makes a novel contribution to the area of tracing requirements to regulatory codes. It highlights the term-mismatch problem and its impact upon traceability, and proposes a ontology-based tracing approach to address this problem. In particular, this approach uses ontology to enhance concepts in the query and therefore make effective associations between query and document. The ontology-based approach holds significant promise for future success in achieving the goal of accurate, automated traceability [27]. Whereas machine learning and web-mining approaches expand or modify query terms without explicit explanations, ontology-based approach aims to add some

semantics which enable higher levels of reasoning. Ontology construction is notoriously effort intensive; however, we have demonstrated the potential viability of leveraging trace links in one project to build ontology that can be reused across multiple projects – thereby recouping the costs involved in the initial ontology construction effort.

On the other hand the semantic information utilized in the ontology-based tracing approach is limited because it ignores the sentence order and still treats each trace query as a bag of words/phrases. In the next chapter, we will focus on how to integrate deeper artifact semantics during the tracing process by considering the specific roles of the domain concepts appeared in software artifacts and the tracing heuristics used by human experts, with the aim of further improving the accuracy of the tracing task.

CHAPTER 3

TOWARDS AN INTELLIGENT DOMAIN-SPECIFIC TRACEABILITY SOLUTION

This chapter presents the work entitled "Towards an Intelligent Domain-Specific Traceability Solution" that was published in the *Proceeding of the 29th IEEE/ACM International Conference on Automated Software Engineering* [58].

3.1 Introduction

In Chapter 2, we proposed an ontology-based tracing technique to break through the barrier of term-mismatch that occurs frequently across software artifacts to be traced. The ontology is effective to expand artifacts with domain related terms. However, it fails to take into consideration the specific roles played by different concepts in the context of the artifacts. Consequently it can not handle the nuances of why two artifacts with similar terms, or even terms related through the ontology, should or should not be linked. Approaches that rely purely on ontology therefore may improve recall but still fail to adequately address the precision problem [85]. In this chapter, we deliver a Domain-Contextualized Intelligent Traceability (DoCIT) system, capable of performing human-like reasoning for the highly focused area of communication and control systems within a transportation domain. While there are non-trivial costs involved in creating a DoCIT system for even a single regulatory domain, this initial outlay must be balanced against the far greater costs of establishing a sufficient set of trace links to demonstrate compliance and safety for such systems. Our experiences of working in this domain have shown that traceability costs for a single project can easily exceed $1 Million (US) [14]. Fortunately, as regulations apply

across multiple projects, the DoCIT knowledge base can be reused in projects that need to comply to the same regulatory codes.

The remainder of the chapter is laid out as follows. Section 3.2 and 3.3 provide an overview of DoCIT and the targeted domain. Section 3.4, 3.5, and 3.6 describe action frame[1], link heuristics, and the knowledge base respectively. Section 3.7 describes the process we used for constructing the initial knowledge base and the way we implemented DoCIT in conjunction with a term-based approach. Section 3.8 describes a series of experiments we conducted to evaluate DoCIT. Finally, Section 3.9 and 3.11 discuss threats to validity and conclusions.

3.2 DoCIT

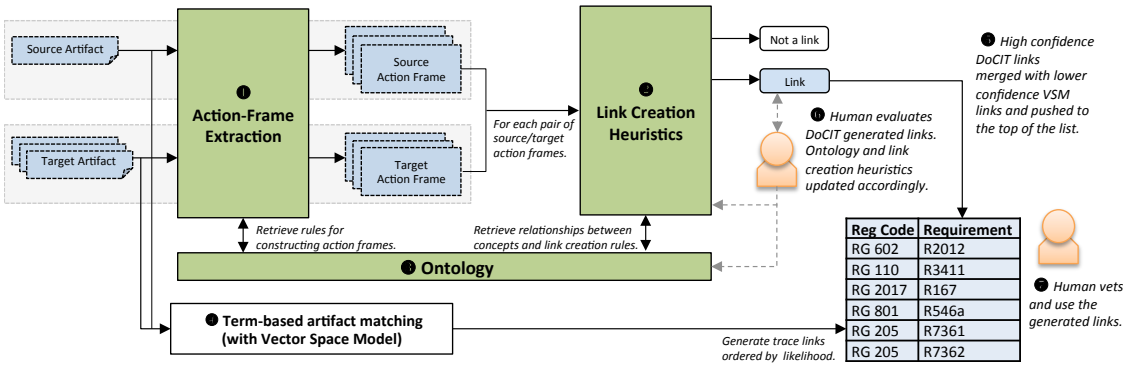


Figure 3.1. The DoCIT process showing its primary components and execution context.

[1]*Action Frame* is referred to as *Action Unit* in the original paper [58].

The various elements of DoCIT are depicted in Figure 3.1. DoCIT takes as input a *source* artifact (e.g. a regulatory code) and a set of *target* artifacts (e.g. system level requirements) to be traced. Trace links are built around *action frames* extracted from source and target artifacts ①. Link heuristics define the conditions under which the properties of two action frames determine that a link should be created between their associated source and target artifacts ②. Both the action frame extraction process and the application of link heuristics are supported by a domain-specific ontology ③.

While DoCIT tends to deliver far more precise trace links than term-based algorithms, it depends on a relatively complete and correct set of action frame definitions, action frame extraction rules, link heuristics, and supporting ontology. We refer to these collectively as the DoCIT **Knowledge Base (KB)** from now on. Given that we cannot guarantee completeness of this knowledge base, especially when tracing new datasets, we deploy DoCIT within the 'safety-net' of the popular, term-based Vector Space Model (VSM) ④. By merging DoCIT generated links with term-based generated ones, we achieve the best of both worlds by pushing the more precise DoCIT links towards the top of the merged list of candidate links. Furthermore, we leverage the VSM similarity scores to create an ordered ranking of DoCIT links ⑤.

Finally, we incorporate human feedback into our process. This facilitates ongoing refinements in the DoCIT knowledge base ⑥, and assigns the human analyst the final task of vetting the generated trace links ⑦.

The work we present in this chapter builds on our previous proof-of-concept work[57]. However, our earlier work relied upon a human analyst to manually map artifacts onto a simple set of seven trace creation heuristics, and utilized a rudimentary ontology. In contrast, the work in this chapter makes very significant advances in moving toward our longer-range goal of complete automation. Knowledge is now represented in a formal ontology language. The novel concept of action frames and their associated extraction rules are introduced, allowing us to automatically parse and analyze non-trivial industrial

requirements. Furthermore, the previous high-level link heuristics are discarded and replaced with 33 heuristics centered around the new notion of action frames. Given an underlying knowledge base, these link heuristics allow us to automate the task of generating a relatively precise set of trace links.

3.3 Transportation Domain

The purpose of this chapter is to demonstrate that a DoCIT solution can be used effectively in a complex regulated domain. We focus our efforts on the communication and control aspects of the transportation domain for several reasons. First, it represents a regulated safety-critical industry for which traceability is required. Second, its projects tend to grow very large with tens of thousands of trace links and therefore the tracing task can be particularly costly and arduous. Finally, we have access to two large industrial datasets from our collaborators. Due to the proprietary nature of the project, we are not able to disclose its exact domain, and therefore all of our examples in this chapter have been disguised under the domain of communication and control for the Driver-Optional Highway System (DOHS). We established a one-to-one mapping between terms in our domain and terms in the DOHS. The DOHS provides environmental controls used by both manned and unmanned vehicles to propose routing, to prevent vehicles from entering accident zones, and to provide a wide range of directives.

The dataset includes 242 System Requirement Specification artifacts (SRS), 963 System Design Description artifacts (SDD), and 583 SubSystem Requirement Specification artifacts (SSRS). Our industrial collaborators provided two validated trace matrices which included 583 trace links from SRS to SDD, and 700 trace links from SSRS to SDD.

3.4 Action Frames

The *action frame* represents a novel and fundamental building block of our approach. In this section we explain what an action frame is, how it is defined, and finally how it is instantiated through a natural language extraction process.

3.4.1 Action Frame Definition

Each action frame is defined in terms of a *verb syntactic* group, a set of *properties*, and a set of *mapping rules* which identify a part of speech to be mapped to each property.

A verb syntactic group represents a group of verbs which belong together because they tend to be *used* in similar ways. For example, the verbs *transmit*, *broadcast*, *indicate*, *upload*, and *deliver* do not mean exactly the same thing, but they do exhibit the same syntactical usage pattern. As a result they also exhibit similar part-of-speech dependencies. For example, we can say that when one of these verbs is found in a sentence, the recipient of the action is normally identified as the noun phrase which is the object of the preposition *to* associated with the verb.

The action frame's properties are then defined in terms of thematic roles [111], which represent a standard set of roles that a noun phrase can play with respect to a verb. While linguistic theories have recognized over 1000 different types of thematic roles, we observed six of them in the data set from which the knowledge base was constructed. These are:

- **Agent:** The initiator of some action, capable of acting with volition, as in "The **DOHS system** shall send..."
- **Theme**: The entity which is moved by an action, as in "...shall send a **message**"
- **Recipient**: The object which receives or gains something as a result of the action, as in "...send a message to a **subsystem**"
- **Instrument**: The object with which or by which the action is performed, as in "...send a message through an **interface**"
- **Location**: The place in which the action is completed, as in "...send a message within the **limits**"

- **Initial Location**: The place in which the action is initiated, as in "...leave the **area**". This role was created to account for sentences containing two locations, as in "transition from A to B".

The work in this chapter is based on these six observed roles, however DoCIT algorithms are extensible to accommodate additional roles as needed.

Finally, the action frame definition provides a set of mapping rules which describe the part-of-speech dependencies that must be mapped to each property. Different definitions use various combinations of properties and dependencies.

For example, in Figure 3.2, an action frame definition is given for a syntactic group related to the set of actions *transmit*, *upload*, *convey*, etc. The definition prescribes the need to identify an *agent*, *theme*, and *recipient* and specifies that these roles are played by the subject (nsubj), direct object (dobj), and prepositional object of *to* (prep_to) respectively.

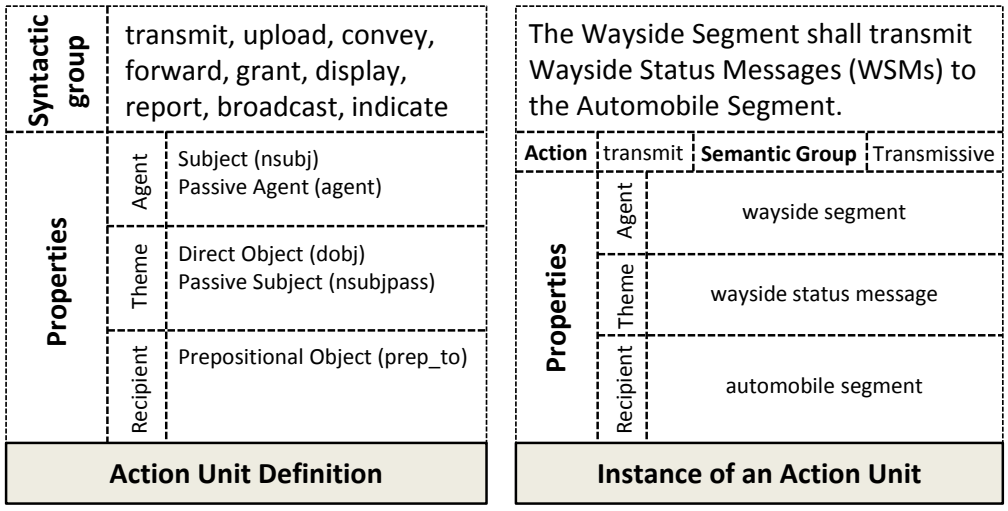


Figure 3.2. A definition of an action frame showing sample properties and an example instantiation

The same figure shows an instantiated action frame. Populated through parsing the artifact "The Wayside Segment shall transmit Wayside Status Messages (WSMs) to the Automobile Segment." The action *transmit* was recognized as a member of the *Transmissive* semantic group, and the noun phrases *wayside segment*, *wayside status message*, and *automobile segment* were mapped to the agent, theme, and recipient properties respectively. We discuss the purpose of the semantic group in the next section.

3.4.2 Instantiated Action Frames

An action frame needs to be identified and extracted automatically from the text in an artifact. The challenging aspect of this task is that artifacts, such as requirements and regulatory codes, are written in natural language, often describe multiple features, carry extraneous information such as rationales, definitions, and other forms of description, often describe complex constraints and conditions, and are not always grammatically well-formed.

Action frame extraction is a natural language processing (NLP) task. We used the Stanford parser for this task because its unlexicalized PCFG Parser [78] can achieve relatively accurate parsing results with fast processing speed. We now provide an overview of the parsing process.

3.4.2.1 Preprocessing

The text from the artifact is first preprocessed by replacing terms such as (/) with *or*, and removing superfluous characters such as (-) which confuse the parsing process. In addition, unhelpful, frequently occurring parts of the sentence, such as *provide [a mechanism] for* were removed through a string pattern matching process. Such patterns have been added to our knowledge base.

3.4.2.2 Parse Tree

The Stanford parser was used to produce a POS (Part-of-Speech) tree as illustrated in Figure 3.3. Each node is labeled with a POS tag. We limited our approach to support only noun phrases (NP) as they have been shown to play the most crucial role in the tracing process [148]. NPs such as *all critical data* can be identified by extracting nodes labeled *NP* in the parsed tree. Phrases which have similar meanings but different structures, such as *signal state* and *state of signal*, are transformed into a standard form using tree pattern analysis and matching techniques.

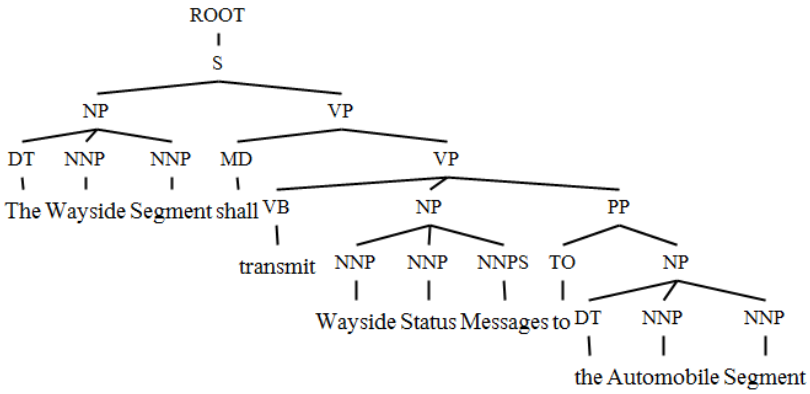


Figure 3.3. Parsed tree of sample artifact

Overall, we processed basic noun phrases, lists of noun phrases, and also prepositional phrases, verb phrases, and clauses found within a noun phrase. From this tree, the Stanford parser generated a dependency set, as depicted in Table 3.1, which provides the information needed to automatically map POS to the properties defined in each applicable action frame definition. Thus, a dependency such as *nsubj(transmit-5, Segment-3)* relates *(Automobile) Segment* as the subject of *transmit*. In general, the NP in the *nsubj* dependency with the

TABLE 3.1

DEPENDENCIES EXTRACTED FROM THE PARSE TREE

det(Segment-3, The-1)	nn(Messages-8, Status-7)
nn(Segment-3, Wayside-2)	*dobj(transmit-5, Messages-8)*
nsubj(transmit-5, Segment-3)	det(Segment-12, the-10)
aux(transmit-5, shall-4)	nn(Segment-12, Automobile-11)
root(ROOT-0, transmit-5)	*prep_to(transmit-5, Segment-12)*
nn(Messages-8, Wayside-6)	

verb *transmit*, along with any other verb in its syntactic group, should be considered the agent of the verb. From the dependency set, we may then extract an instance of the action frame from an artifact in an automated fashion as each relevant dependency links a verb to nouns that get mapped onto action frame properties.

3.4.2.3 Semantic Verb Groups

There is one final step in the action frame instantiation process. While properties and their mappings are driven by the *usage* patterns of the verb, the link creation process is driven by their *semantics*. The final step in the process therefore involves looking up the semantic group of the verb. Our current knowledge base includes 24 distinct semantic groups such as *Affirmative*, *Calculative*, *Evasive*, and *Receptive*. While standard semantic groupings do exist (i.e. in WordNet), they tend to focus on quite commonly used verbs and do not include many of the verbs found in our transportation domain. Furthermore, in many cases, no appropriate grouping exists in which a new verb can be added. We therefore developed our own semantic groupings following the process described in Section 3.7.1.

The end result of the extraction process is that each artifact is represented by one or more instantiated action frames. It is these action frames which are then used in the trace

link generation process.

3.5 Link Heuristics

Trace links are created by comparing two action frames extracted from a source and target artifact respectively. Currently, a trace link is established between the source and target artifact whenever a relationship is established between a pair of their action frames. However, in future implementations we plan to take all pairs of action frames for a source and target artifact into account simultaneously in order to reason more effectively about conditions and constraints.

3.5.1 Link Heuristic Definition

Each link heuristic provides a set of guidelines for evaluating whether two action frames of specified semantic groups should be linked or not. For this reason, most heuristics are named according to a semantic pair, for example: transmissive-receptive, motive-permissive, or administrative-inceptive. The transmissive-receptive heuristic can only be applied when one action frame is of type transmissive, and the other is of type receptive, and so on. Currently, the only exception is the *basic* heuristic, which can be applied whenever two action frames share the same semantic group.

Link heuristics also specify rules for comparing the action frame properties. For example, a heuristic might specify a rule that the *themes* of two action frames must be the same, or that the *agent* of one action frame must match the *recipient* of another. Such matches can be either *exact* or *hierarchical*. An exact match means that two elements are the same or equivalent, while a hierarchical match means that two elements are related through the is-a or compositional hierarchy of the ontology. For example, a *Wayside Interface Unit* is part of the *Wayside Segment*. Such hierarchical and compositional relationships are captured in the domain ontology which we describe in Section 3.6.

3.5.2 Sample Link Heuristics

We illustrate our approach with three different link heuristics. Each is defined in terms of its semantic groups, as well as the rules by which the elements must match, where an element is defined as the terms assigned to a property in the action frame. For example, *WIU* might be the element assigned to a theme property.

There are currently four types of matching criteria (MC) defined as follows for a pair of elements:

- **MC 0** : If both elements are present, then they must match (exact or hierarchical). However, if one or both of the elements are missing, the MC can be ignored.

- **MC 1** : If one element is present, the other must be present too and must match (exact or hierarchical). If neither element is present, the MC can be ignored.

- **MC 2** : Both elements must be present and must match (exact or hierarchical).

- **MC 3** : Both elements must be present and must match exactly.

3.5.2.1 Basic Link

The basic heuristic is applicable when the semantic group in both action frames are the same. A link is established if and only if *agents*, *recipients*, and *instruments* are matched at MC level 0, and *themes* and *locations* are matched at MC level 1. We provide the following illustrative example:

A1: Any critical failure during the Disengaged Mode will force the OBM to enter the Failed Mode.	
Action	enter
Semantic Group	Motive
Themes	obm
Location	fail mode

A2: The VehicleGuard DOH System shall have a Failed Mode where the Automobile segment transitions if a vital hardware or software failure is detected.	
Action	transition
Semantic Group	Motive
Themes	automobile segment
Location	fail mode

In this case all the criteria for the heuristic are met. Both action frames belong to the *Motive* semantic group. The OBM (i.e. Onboard Module) is part of the automobile segment; therefore, the themes match at the hierarchical level. Finally, the two locations exhibit an exact match. Therefore a trace link is established.

3.5.2.2 Transmissive-Receptive

This heuristic applies when the semantic group in one action frame is *Transmissive* and the other is *Receptive*. A link is established if and only if *agents*, *recipients*, *instruments*, and *locations* match at MC level 0, and *themes* match at MC level 2. We provide the following example.

A1: The OBM shall support reception and decomposition of Wayside Status Messages (WSM).	
Action	reception
Semantic Group	Receptive
Recipients	obm
Themes	wayside status message

A2: The Wayside Segment shall transmit information to the Automobile Segment in the form of Wayside Status Messages (WSMs).	
Action	transmit
SemanticGroups	Transmissive
Agents	wayside segment
Recipients	automobile segment
Themes	wayside status message

In this case the semantic groups are *Transmissive* and *Receptive* and so the heuristic is applicable. Checking the property pairs, we see that *agents, instruments*, and *locations* are missing and therefore ignored. Furthermore, the *recipients* property is present in both cases, and *OBM* stands for *Onboard Module*, which is a part of the *Automobile Segment* and therefore hierarchically linked through the ontology. Finally, both action frames contain the same theme of *Wayside Status Message*. As a result, a link is created.

3.5.2.3 Motive-Permissive

This heuristic applies when the semantic group in one action frame is *Motive* and the other is *Permissive*. A link is established if and only if *locations* match at MC level 3, and *themes* match at MC level 0. We illustrate this with a final example.

A1: The Cut-Out mode is entered automatically from the self-test mode.	
Action	enter
SemanticGroups	Motive
Location	Cut-Out Mode
InitialLocation	Self Test Mode

A2: The VehicleGuard DOH System shall be forced into cut-out mode by selecting the cut-out mode setting on the sealed mode switch on the OBM unit.	
Action	force
SemanticGroups	Permissive
Themes	VehicleGuard DOH System
Location	Cut-Out Mode

In this example, the semantic groups are *Motive* and *Permissive*; *location* properties are an exact match, and as the *theme* property is present on only one side, it is ignored. A link is therefore created.

Our current knowledge base includes 33 link heuristics, each of which is summarized in Figure 3.4.

3.6 Ontology

Both the action frames and link heuristics are supported by an underlying ontology. Without the ontology we would be unable to understand conceptual relationships between different terms and would therefore be constrained by the same problems as more basic term-matching approaches.

A knowledge base is an information repository used to collect, organize, and search data [75, 132]. Many KBs are built in a manner that supports automated deductive reasoning. Such KBs are composed of a set of data that includes basic terms that define the

Semantic Groups	Agent	Theme	Recipient	Instrument	Location	Init Loc.	$Agent_1$/$Recip._2$	$Recip._1$/$Agent_2$	$Instrum._1$/$Theme_2$	$Init\ Loc._1$/$Loc._2$	$Loc._1$/$Theme_2$	$Agent_1$/$Theme_2$
Basic (Groups must match)	❶	❶	❶	❶	❶	❶						
Affirmative-Inclusive	❶	❷										
Calculative-Inclusive	❶	❷										
Descriptive-Transmissive		❷										
Enforcive-Calculative (1)	❷								❷			
Enforcive-Calculative (2)	❶	❷	❶	❶	❶							
Enforcive-Enforcive	❷								❷			
Enforcive-Receptive		❷					❶					
Enforcive-Regulative (1)	❶										❷	
Enforcive-Regulative (2)	❶	❷										
Inceptive-Inceptive	❶	❷		❶	❶							
Inceptive-Receptive		❷					❶					
Inceptive-Transgressive (1)		❷			❸							
Inceptive-Transgressive (2)					❸							❷
Inspective-Respective							❶		❷			
Motive-Motive					❸							
Necessitative-Receptive							❶		❷			
Permissive-Motive		❶			❸							
Permissive-Permissive					❸							
Preservative-Inceptive	❷	❷										
Preservative-Preservative	❶	❷	❶	❶	❶	❶						
Receptive-Receptive		❷						❸				
Receptive-Transmissive (1)		❷					❸					
Receptive-Transmissive (2)	❶	❷	❶	❶	❶							
Regulative-Inclusive	❶	❸	❶	❶	❶							
Regulative-Transmissive		❷					❶					
Transgressive-Motive		❶			❷							
Transgressive-Permissive					❸							
Transmissive-Administrative		❶					❷					
Transmissive-Enforcive		❶					❷					
Transmissive-Inceptive	❷	❷										
Transmissive-Permissive		❷						❸				
Transmissive-Transmissive	❶	❷										

❶ = If both elements are present, they must match (exact or hierarchical).

❶ = If one element is present, the other must also be, & match (exact or hier.)

❷ = Both elements must be present and must match (exact or hierarchical)

❸ = Both elements must be present and must exhibit an exact match.

Figure 3.4. Trace Link Heuristics specified in terms of semantic pairs and required matches.

vocabulary of the domain, and a set of sentences that describe the relationships between those terms. The data is often represented in the form of an ontology in which knowledge is constructed using logical operators such as *AND* and *OR*, as well as *implication* and *negation* operators to build complex ideas from more primitive concepts [75].

Our KB performs two primary functions for tracing purposes: using the knowledge in the KB to *compare* two artifacts and *mapping* elements of software artifacts to concepts in the ontology. We also store syntactic group information and semantic group information in the knowledge base so that given a verb, we can determine which groups it belongs to.

The DoCIT KB was created using OWL 2 Web Ontology Language, a formally defined ontology language developed by W3C for the Semantic Web [69]. OWL 2 is capable of representing complex knowledge about things, groups of things, and relationships between those things. OWL 2 ontologies provide classes, properties, individuals, and data values, all of which are stored as Semantic Web documents. We created our initial KB using Protégé, an open source ontology editor [135], and saved it in an OWL (.owl) format for use during the tracing process.

3.6.1 Knowledge Structure

The KB contains general concepts, which are expected to be reusable across multiple domains, and also domain-specific knowledge which is designed solely for use in a specific regulatory domain, e.g., communications and control for the transportation sector of our case study.

The general section of our KB includes facts about *verbs* and *verbs used as nouns*, such as *reception* or *integration*. These are represented as individual elements in the ontology. Every identified verb belongs to one or more semantic groups representing its functionality. For example, the verb *enter* is used in two functionally distinct ways across the set of observed artifacts: first, as in "The system should enter the Failure Mode when a vital error occurs" (the **Motive** group), and second, as in "The driver should be able to enter the

current date into the system from the HMI" (the **Transmissive** group).

Each verb is also categorized under one or more syntactic groups based on how it interacts with the surrounding noun phrases. Reexamining the previous example, we see that the *Failure Mode* is a location that the *system* should *enter*, while the *current date* is something the *driver* should *enter*, which gets received by the *system*. Despite having similar syntactic distributions, the arguments of *enter* in each case play different roles in determining the main function contained in each sentence.

The second part of the ontology describes domain-specific taxonomic relations between objects, such as names of systems, subsystems, operating modes, and message types used by components of the system.

3.6.2 Querying the KB

For an ontology to be useful, it needs to provide a mechanism for supporting queries and returning relevant information. The DoCIT ontology therefore provides a query function which can accept either a single term or a noun phrase and return a set of related concepts as well as a set of relations. For example, if we issue a simple query for the term *update*, the KB returns the information that *update* is an action and belongs to the *regulative* semantic group and syntactic group # 3 (as syntactic groups are labeled by ID only).

Similarly, a query issued for the noun phrase *Power-On Self Test* returns a rich set of hierarchically related terms, among other things, specifying that the Power-On Self Test is a kind of relay test, which in turn is a kind of self-diagnostic test. This information is critical for enabling traceability between artifacts which may use different terms that are related through a hierarchy of concepts.

3.7 DoCIT in Practice

Deploying DoCIT required construction of the knowledge base and development of a functioning prototype.

3.7.1 Building a Knowledge Base

Our current version of DoCIT requires the presence of a knowledge base composed of action frames, link heuristics, and a domain ontology. In this section, we describe the systematic approach we followed to build the knowledge base needed to support automated trace link generation in our domain of study. We worked closely with a domain expert from our collaborating industry to ensure that our reasoning, and the subsequent results, were well-founded. Articulating these steps is an important precursor towards complete automation of ontology building for tracing in a given software engineering domain.

Step 1. Create Verb Syntactic and Semantic Groups: The Stanford parser was used to identify verbs and verbs represented as nouns. For each identified verb we manually checked whether an appropriate syntactic group and semantic group existed. To identify relevant syntactic groups we asked, "Is there an existing syntactic group which contains verbs conveying similar behavior?" To identify relevant semantic groups we asked, "Is there an existing semantic group which contains verbs conveying similar meaning?" If not, we created a new syntactic and/or semantic action group.

Step 2. Create Action Frame Construction Rules: For each new syntactic group, we created a set of action frame construction rules. These rules specify the relevant properties (i.e. thematic roles such as location, agent, etc.) and the appropriate dependent POS which should, if present, be associated with the property.

Let us assume that no existing syntactic group exists for the verb *send*. First, we run the Stanford parser to identify verbs and their dependencies. We then inject a human reasoning step. For example, we might determine that the action "send" requires a direct object, i.e.

"send what?" Furthermore, this direct object is generally an entity that is moved by the action. We therefore add "theme" as a property to the action frame and specify that the *direct object* serves this role. A similar process is used to add other relevant properties and their associated parts-of-speech.

Step 3. Create Link Heuristics: Link heuristics are built around pairs of semantic groups. The occurrence of a previously unseen pair of semantic groups will trigger the need for a new link heuristic to be created through analyzing the likely thought process of a human analyst. First we determine the rationale for the link, and secondly we formulate this in terms of the thematic roles. We frequently engaged our industrial collaborator in this process so that he could confirm or refute our rationales for each link.

It is worth noting that our current implementation takes a rather greedy approach to constructing the various rules, heuristics, and concepts. Each of these elements is added to the knowledge base if it benefits the currently evaluated artifact and/or trace link. While, we considered implementing a "do no harm" policy, i.e. not adding new information if it harmed previous trace links, we rejected this idea in favor of a more global optimization to be explored in future work.

3.7.2 Integration with Term-Based Approach

As previously explained, we implemented DoCIT in conjunction with the Vector Space Model (VSM) [65]. We used the standard VSM algorithm as introduced in Section 2.2.2.

3.7.2.1 Merging Results

To use VSM in conjunction with DoCIT, the DoCIT approach is first applied to the source and target artifacts to produce a list of candidate links referred to from now on as candidate DoCIT-links. VSM is then applied to the same source and target artifacts to produce a ranked list of candidate VSM-links. DoCIT tends to produce fewer, but more accurate links than VSM. Therefore, a merged list of candidate links is created by placing

DoCIT links at the top of the list ordered according to their VSM rankings, and then adding all remaining VSM-Links in their original order to the bottom of the merged list.

3.7.2.2 Learning New Facts and Rules

The process of discovering and specifying link heuristics is ongoing, and therefore any knowledge base needs the ability to grow and expand over time as new information, acronyms, or sentence structures are seen for the first time. We therefore developed a GUI tool which incrementally displays links which were missed by DoCIT but ranked highly by VSM. These false negatives (from DoCIT's perspective) provide an ideal opportunity for eliciting knowledge from the domain expert who is performing the tracing task. The source and target artifacts of the link are parsed and action frames are displayed. The tool then provides a GUI to help the user to construct new link heuristics.

3.8 Evaluation

To evaluate DoCIT's ability to support trace link generation we designed and executed four different experiments labeled E1 to E4. The first evaluated DoCIT's ability to correctly identify and retrieve action frames from both the SRS and SDD datasets. The second and third experiments evaluated DoCIT's ability to generate trace links correctly with varying degrees of support from the knowledge base, and finally the fourth one analyzed the efficacy of the link heuristics. An alpha level of 0.05 was used for all statistical tests.

3.8.1 E1: Action Frame Extraction

Our DoCIT solution is intended to work in complex software systems with real requirements. The first experiment was therefore designed to address the research question **(RQ 1)**: "Can DoCIT extract action frames correctly and consistently from complex industrial requirements?"

3.8.1.1 Experimental Design

To answer this question we used DoCIT to extract action frames from 20 randomly selected SRS artifacts and 20 randomly selected SDD artifacts. Each action frame was then evaluated by two researchers in our team. Action frames that were correctly extracted according to the definitions in the action frame were classified as **Correctly parsed**. In addition, when the mapping was not successful, we classified the failure causes into the following three categories: (1) **Parser errors** i.e. the parser generated an incorrect tree structure or incorrectly assigned part-of-speech tags and as a result the retrieved action frame was incorrect, (2) **Convoluted sentence** i.e. the sentence was very confusing and produced multiple action frames. As a result each action frame was too sparse to support effective link creation. An example of a convoluted sentence is "The system should provide the functionality by which the subsystem under its control can perform.....", (3) **Missing verb** i.e. a verb identified by the parser cannot be found in the set of verb syntactic groups.

3.8.1.2 Results and Analysis

Results are reported in Figure 3.5. The SDD sample produced 51 action frames, while the SRS dataset produced 35, reflecting the fact that SDD artifacts in this dataset tend to contain longer, more complex descriptions than SRS ones.

Of the 51 SDD action frames, 39 were correct, 8 failed due to parser errors, 3 due to convoluted sentences, and 1 due to incomplete verb knowledge in the knowledge base. Of the 35 SRS action frames, 34 were correct and the single failure was due to a parser error. Action frames were constructed for each identified verb, and no missing ones were observed.

These results show that DoCIT was able to correctly extract 76.5% of the links from SDD and 97% from SRS. The main cause of failure in both cases was parser errors, which can be addressed in future work through adding additional pattern matching rules to the knowledge base.

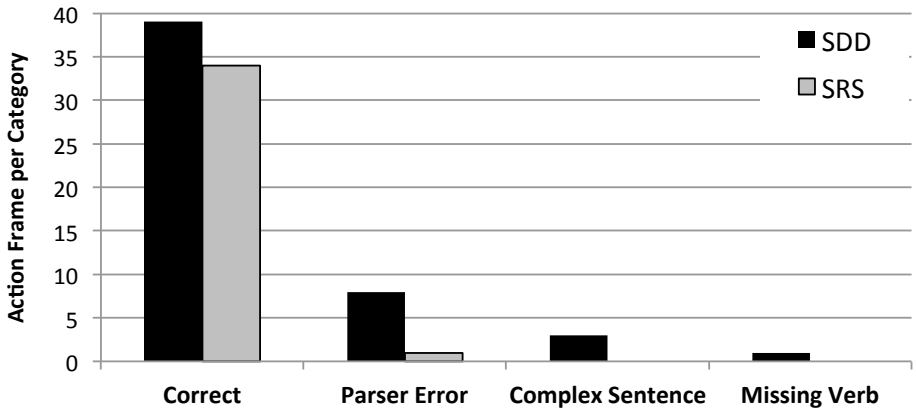
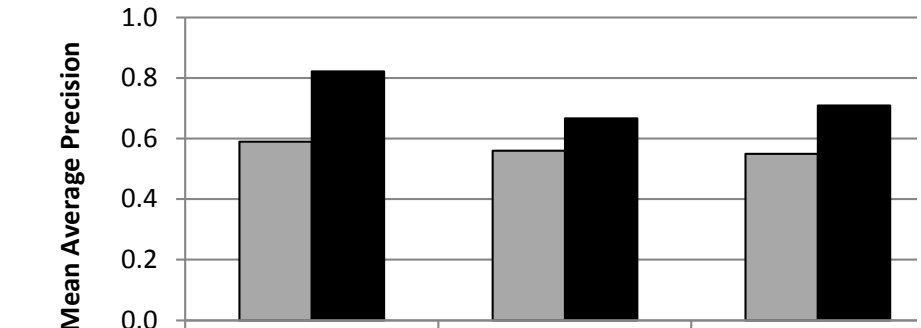


Figure 3.5. Analysis of generated Action Frames



	E2: SRS1-SDD	E3: SRS2-SDD	E3: SSRS-SDD
VSM Only	0.590	0.560	0.550
VSM+DoCIT	0.822	0.667	0.709

Figure 3.6. MAP by query and link for each of the three datasets in Experiments 2 and 3.

3.8.2 E2: When KB Is Relatively Complete

The second experiment evaluated DoCIT's ability to generate trace links when the knowledge base was customized for the dataset. This represents an ideal scenario in which the knowledge base provides relatively complete coverage of the concepts in a dataset. We addressed the research question (**RQ 2**): "Given a relatively complete knowledge base can

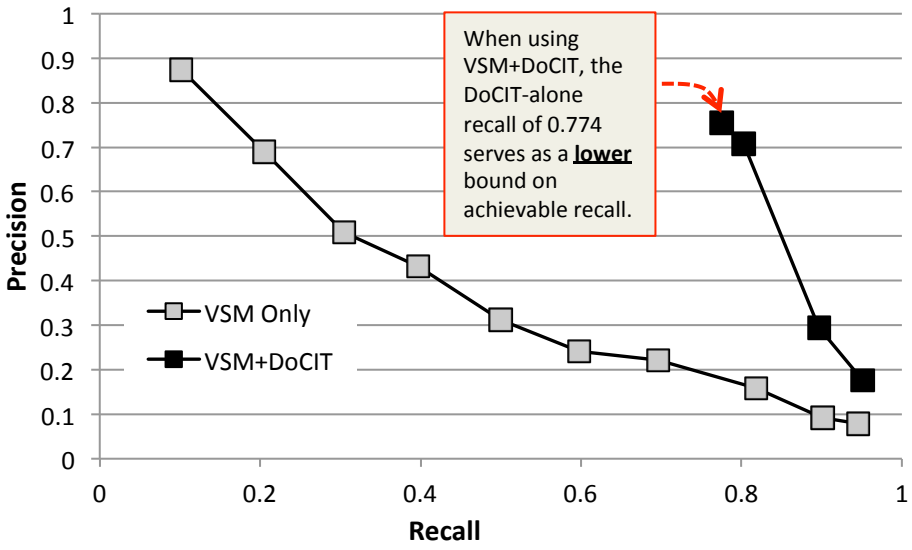


Figure 3.7. Precision versus Recall for Experiment #1

DoCIT return an accurate set of trace links?"

3.8.2.1 Metrics

Results were evaluated using three standard metrics. Recall measures the fraction of relevant links that are retrieved while precision measures the fraction of retrieved links that are relevant. A third metric, Mean Average Precision (MAP), measures the extent to which correct lists are returned at the top of a ranked list. The formula for calculating of MAP can be found in Section 2.2.3. MAP requires an ordered ranking of the links and so is only reported for results that include VSM. As our implementation of MAP examines all correct links it achieves recall of 1.0.

3.8.2.2 Experimental Design

For this experiment we randomly selected 31 SRS artifacts and the complete set of SDDs with links from any SRS artifact. This resulted in 241 SDD artifacts, producing a

total of 7,471 potential links. Of these 205 were included in the trace matrix created and provided by our industrial collaborators. We refer to this subset of the dataset as *SRS1*.

Starting with the initial knowledge base, two researchers on the team systematically evaluated the 205 trace links, and following the greedy process prescribed in Section 3.7.1, systematically added additional concepts to the knowledge base and created new link heuristics. Once each link had been examined and the knowledge base expanded, trace links were generated for the 31 SRS using both VSM and DoCIT.

3.8.2.3 Results

MAP scores are reported in Figure 3.6 and show that VSM alone returned a MAP of 0.59 compared to 0.82 when used in conjunction with DoCIT. A Wilcoxon Signed Rank test showed a significant difference when the VSM MAP score was compared against VSM+DoCIT MAP scores for each query at $p<0.001$. Figure 3.7 reports precision at various recall levels and also shows a marked improvement in accuracy when VSM+DoCIT is used instead of just VSM. For example, at recall levels of approximately 0.8, VSM returns precision of only 0.17 compared to 0.71 for VSM+DoCIT. This is a non-trivial improvement in accuracy and supports our overall conjecture that a more intelligent domain-specific system has the ability to far outperform a term-based approach such as VSM. We chose to not compare our approach against alternate ontology-based tracing techniques as published results have shown inconsistent improvements over VSM [85].

3.8.2.4 Towards Even Greater Accuracy

Our goal is for DoCIT to ultimately achieve MAP scores close to 1.0 (i.e. to place all of the correct links close to the top of a ranked list) when the underlying knowledge base is complete. However, while DoCIT significantly improves MAP and precision, there is still room for improvement. We therefore analyzed both false positives and false negatives and make the following observations listed in order of their impact: (1) Errors of commission

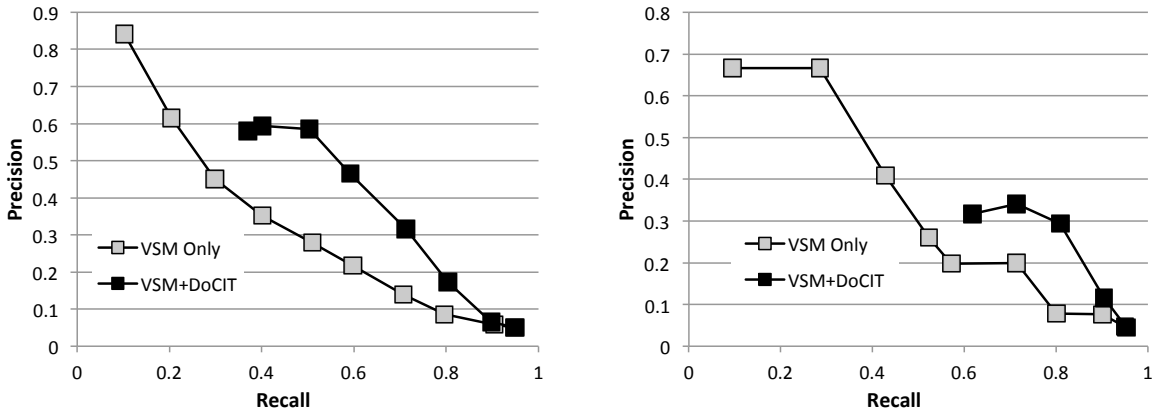


Figure 3.8. Results from Datasets SRS2 and SSRS with incomplete Knowledge

(i.e. false positives) were primarily caused by the fact that DoCIT generates a link if even one pair of action frames are related by the link heuristics. This approach fails to filter out links that should have been rejected when leading or trailing conditions do not match. (2) Errors in omission (i.e. false negatives) were primarily caused by incomplete knowledge in the knowledge base. (3) A few errors were caused by parsing mistakes as previously discussed in Section 3.8.1. All three error types will be addressed in the next phases of our work.

3.8.3 E3: When the KB Is Incomplete

Software projects are rarely static. New artifact types are added, and new domain concepts including syntactic and semantic verb groups may be introduced as the project evolves. As a result we can never expect, and certainly not guarantee, a complete knowledge base. This introduces the research question **(RQ3)**: "When a knowledge base is incomplete does DoCIT still lead to improvements in trace accuracy?"

3.8.3.1 Experimental Design

We conducted an experiment against two different datasets for which the KB provided only partial coverage.

For the first dataset we randomly selected 31 different SRS artifacts (i.e. not ones used in the previous experiment) and the same complete set of 241 SDDs. This produced a total of 7,471 potential links of which 157 were marked as correct by our industrial collaborators. We refer to this subset of the dataset as *SRS2*. Because the initial KB was constructed through examining a different subset of SRS-SDD trace links (i.e. SRS1), we would expect partial knowledge coverage of concepts in SRS2 in the KB.

For the second dataset we selected 10 SSRS artifacts and traced them to the same 241 SDDs. SSRS represented a completely new type of artifact from the transportation dataset that had not been previously been used to build the Knowledge Base. There were a total of 21 correct trace links out of the potential set of 2410. We refer to this dataset as *SSRS*. As SSRS represents an entirely new artifact type that was not used in the KB construction, we expect less knowledge coverage than for SRS2.

The tracing experiment was conducted in the same way as the previous one, with the important difference that no new information was added to the knowledge base. Traces were generated based purely upon previously defined facts in the knowledge base.

3.8.3.2 Results and Analysis

Results are also reported in Figure 3.6. In the case of SRS2, MAP increased from 0.560 to 0.667 and for SSRS it increased from 0.550 to 0.709. Results for SSRS were better than we had anticipated because DoCIT improved five of the ten queries, and in three of these cases returned perfect MAP scores. A Wilcoxon Signed Rank test showed a significant difference for SRS2 at $p = 0.013$, but did not show significance for SSRS at $p = 0.056$. Improvements in precision vs. recall were not as marked as the improvements for SRS1, however, as depicted in Figure 3.8. At fixed recall values precision was markedly

higher for VSM+DoCIT than for DoCIT alone in both datasets. We therefore conclude that even though DoCIT's knowledge base was incomplete with respect to the two datasets, the quality of the links still showed marked improvements.

3.8.4 E4: Greedy Link Heuristic Construction

Given our greedy approach to the construction of link heuristics, we explored the research question **(RQ4)**: "Are all existing link heuristics globally effective?". To answer this question we simply counted the number of times each heuristic was applied for datasets SRS1 and SRS2, versus the number of times it resulted in a correct link. Results show that 51% produced correct links with 100% accuracy, an additional 35% produced correct links with at least 70% accuracy, and the remaining 14% produced correct links with at least 33% accuracy. Even in the worst case of 33% accuracy, they tend to outperform the accuracy of term-based retrieval approaches. Nevertheless this needs further investigation and global optimization will be addressed in future work.

3.9 Threats to Validity

Our results show that integrating DoCIT with VSM led to significant improvements in accuracy of the generated trace links, especially when an adequate knowledge base was present. Construct validity threats were primarily addressed through using utilizing metrics broadly accepted to differentiate between high and low quality links [65]. The generated links were compared against a golden-answer set provided by our industrial collaborators. In cases where additional links were identified as potentially valid, they were reviewed by the industrial collaborators, and when deemed correct, were added to the answer set. Nevertheless, it is possible that additional correct links exist but were not found. Similarly internal validity was addressed through conducting a controlled experiment to compare results obtained using VSM alone versus VSM plus DoCIT.

The major threat of our study focuses around external validity which speaks to the

generalizability of the approach. Our end goal is to build an intelligent traceability system that can be reused across a very narrow set of systems in the same regulatory domain. However, given the significant time and effort it took to obtain the datasets, to compare alternate techniques and ultimately develop a solution based on action frames and link heuristics, to build a domain ontology, and then to carefully validate results, we were only able to conduct our study for a single project in a highly focused domain. The two primary threats to generalizability are (1) whether our action frame extractor will work on a variety of artifacts and styles, across other projects, and in other domains, and (2) whether the link heuristics can generalize across other projects. On the other hand, we do not yet claim generalizability. The goal of this research was to demonstrate that an intelligent traceability solution could significantly improve trace results in a case project. We leave the task of generalizability to future work.

3.10 Related Work

Related work falls under two major categories of transforming unstructured text into formal specifications and use of ontology to support the tracing process.

Previous researchers, in particular Breaux, Anton, et al. have developed techniques for transforming natural language text into formal or semi-formal representations [18, 99]. Their focus has been on the areas of security, privacy [18], and accessibility [17] and has emphasized elements such as *rights*, *obligations*, and *permissions*. Their work evaluated compliance of industrial requirements to accessibility standards by extracting requirements from the standards using an approach they refer to as the frame-based method, and then performing a gap analysis between the two sets of requirements [17]. While they have made some attempts to automate the extraction of various elements from text, their approach is primarily manual in nature, and does not begin to address the issues of automating the traceability process.

Spanoudakis et al. [134] developed a rule-based approach for generating traceability

relations between artifacts, however their rules were primarily syntactical in nature and did not attempt the kind of semantic analysis performed by DoCIT.

Several researchers have previously explored the use of ontology for tracing purposes. Most of these works used ontology to augment the basic approach. We have discussed their strength and limitations in Section 2.7.3. However, unlike DoCIT, their approach did not attempt to establish semantically-aware link heuristics.

Ontologies have been used in the requirements domain for several different purposes, and several techniques therefore exist for extracting an ontology from a requirements specifications [45, 44, 109, 5]. We have proposed a semi-automated approach for generating ontology from domain document collections and existing software trace links [56]. We plan to examine fully automated ontology extracting techniques in our future work.

3.11 Conclusion

The work described in this chapter has made steady progress towards tackling the significant challenge of automating the creation of trace links in a safety-critical, communication and control domain. We successfully demonstrate that within a large industrial project, given a knowledge base with sufficient coverage of the concepts, DoCIT is able to significantly improve the accuracy of generated trace links. These results support our earlier conjecture that intelligent traceability provides a viable approach for overcoming the term-mismatch barrier of existing tracing techniques and ultimately of delivering automated tracing techniques that can be effective for industry.

In the next chapter, we will introduce how we enhance DoCIT with the ability of explaining its reasoning process to the user in natural language as trace link rationales. During the trace link evaluation stage, this information can greatly improve the efficiency when human tracing experts scrutinizing the trace links generated by DoCIT. It can also be especially useful in the scenario when the trace link users are different from the link creators. It presents an initial effort towards preserving and utilizing trace link rationales.

CHAPTER 4

TRACE LINKS EXPLAINED: AN AUTOMATED APPROACH FOR GENERATING RATIONALES

This chapter presents the work entitled "Trace Links Explained: an Automated Approach for Generating Rationales" that was published in *the Proceeding of the 23rd IEEE International Conference on Requirements Engineering* [59].

4.1 Introduction

In Chapter 3, we introduced DoCIT, a tracing solution that utilizes rich semantic information in its link evaluation mechanism. Once the link is created and confirmed by the human user, only a binary decision is recorded as the result of this tracing process, i.e. link or non-link between two artifacts. This is also the case for the manual tracing method during software development process. Very often the reason behind a trace link is hidden in the mind of the engineer who created it. Other stakeholders utilizing the link to perform future tasks may be unable to understand the logic of the link. As a result, potentially critical knowledge is lost, and the overall quality of the software system may suffer.

In some cases, a traceability relationship that exists between two artifacts is reflected in their textual descriptions. Drawing on an example taken from the Medical Infusion Pump (MIP) domain, the requirement "The PCA pump shall maintain an electronic *event log* to record each action taken by the pump and each event sensed of its environment" traces to the design specification "The event logger thread *records all actions or events* for later review or audit." In this case, the two related artifacts both refer to *events*. Furthermore,

one talks about *maintaining an event log*, while the other talks about *record(ing) all events*. To most people, the rationale for the link would be self-evident. However, in other cases, the relationship is less obvious. Consider the requirement "The estimate of remaining battery energy must be accurate to within Xbtty = 25%," which traces to "The power control component is responsible for switches between battery-backup and mains supply, and detects anomalies like voltage out-of-range." In this case the relationship between the artifacts is inferable, but less obvious. The trace exists because, among other things, the ability to switch effectively between power supplies requires the system to be capable of accurately measuring the remaining battery energy.

Consider one more system requirement for a Driver-Optional Highway System (DOHS) that states, "The Highway Wayside Segment shall monitor signal, road work directive, and hazard detector information from field devices", and the related system design artifact that states, "During lamp-out conditions the WIU shall send the current state of the highway signal." In order to establish or assess this trace link, it is necessary to understand that: (1) WIU is an acronym of Highway Wayside Interface Unit; (2) The current *state of the highway signal* is represented as *data*; (3) WIUs are connected to field devices; and (4) The monitoring of field devices requires them to send data. Without this domain knowledge, the safety assessor may not be able to understand the logic behind the established trace link.

Given that trace links will be used by diverse stakeholders, it is important to provide rationales that clearly explain the meaning of each link. For example, certifiers of safety-critical products inspect artifacts and their associated trace links in order to assess whether the manufacturer's safety claims for the product are supported [94]. They therefore need to understand the logic behind each trace link – despite having only partial knowledge of the system under development. Trace link rationales make tacit domain knowledge explicit and thereby help to fill this knowledge gap.

It stands to reason that just as trace links can be created manually or in an automated

fashion, so can trace link rationales be specified by humans or generated by machine. Hull and Dick proposed *Rich Traceability* as a means of explicitly capturing satisfaction arguments between requirements and design, thereby documenting the rationale for a link [72]. Unfortunately, it is time-consuming to create and to maintain rationale documentation manually. Other researchers, such as Balasabrumanian et al. [117] and Zisman et al. [134] have proposed extensive traceability meta-models which not only define the artifacts to be traced, but also the semantics of the links between them. However, while these approaches attach semantic labels to links, they fail to *explain* the purpose of individual links. In this chapter we therefore propose a new technique for generating trace link rationales in an automated way. We build upon our prior work in utilizing domain-specific knowledge bases to support traceability [57, 58] by augmenting the knowledge base with *rationale patterns* that are used to generate facts that provide a logical foundation for the trace link.

4.2 Trace Link Rationales

Rich Traceability, proposed by Hull and Dick, makes use of textual rationales and propositional logic to construct satisfaction arguments between pairs of linked artifacts. As such, it provides an explicit rationale for the trace link, explaining perceived conceptual gaps [72]. Instead of simply representing a trace link in a standard traceability matrix, the link is augmented by a satisfaction argument. The argument may include definitions, relationships between terms, environmental assumptions, and rationales describing the reason for specific design decisions. Arguments can be *single-* or *multi-layered*, depending on their complexity.

For example, consider the requirement shown in Figure 4.1. The logical gap between requirement RS.4.0(3) and the two related design descriptions is filled with rationales describing accepted standards for issuing warnings in medical devices and a definition for *basal rates* that explains the connection to pump flow rates. In Figure 4.2 we provide a second example, this time taken from the DOHS domain. In this domain, many of the

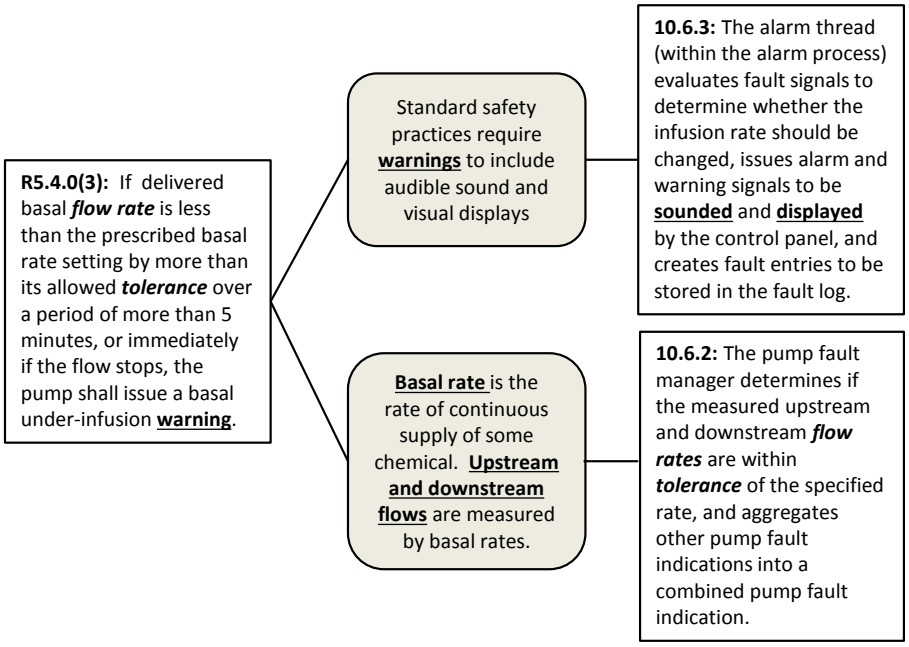


Figure 4.1. *Rich Traceability* showing rationales for two trace links in the Medical Infusion Pump example

trace links can be interpreted only with expert knowledge. Four domain facts are shown. Together, these explain the trace link that exists between requirement SRS-459 and design artifact SSD-682.

According to Hull et al., the effort to establish and maintain the satisfaction arguments is not trivial. They claim that in their Network Rail Project a team of two to five requirements engineers gave dedicated effort to the creation and maintenance of approximately 500 satisfaction arguments over a period of three years [72]. Given the significant effort required to manually create and maintain trace link rationales, we need to look for automated solutions.

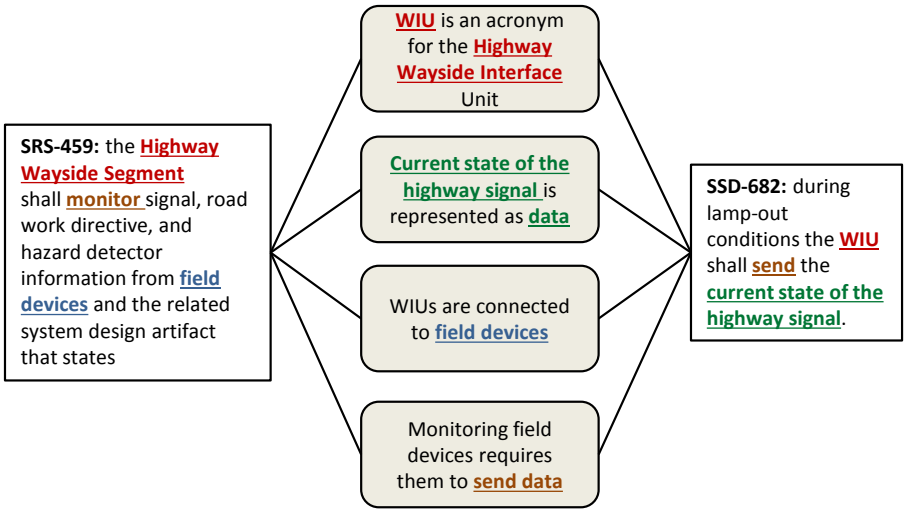


Figure 4.2. *Rich Traceability* showing multiple rationales that contribute to explaining a single link in the DOHS domain

4.3 Rationale Generation

In Chapter 3, we introduced DoCIT, a domain-specific intelligent traceability solution [57]. DoCIT overcomes the term-mismatch problem by utilizing a domain ontology to understand synonymic, generalization, and compositional relationships between domain concepts. It applies natural language processing (NLP) techniques to match terms and phrases onto concepts in the ontology. Mapping rules are then used to construct intermediate representations called *action frames*, and link heuristics are used to establish trace links by matching action frames across source and target artifacts. One example that two matched action frame are illustrated in Table 4.1, along with their original artifacts. In this chapter, we propose an extension to DoCIT that enables it to *explain itself* by generating trace link rationales.

Using DoCIT to generate rationales for trace links is equivalent to explaining the reasoning process DoCIT used (or would use) to establish those links. Rationales are strongly associated with link heuristics and are designed to explain the reason for the link in a way

TABLE 4.1

TWO DOCIT ACTION FRAMES LINKED VIA A HEURISTIC MATCHING PROCESS

A2: The OBM shall support reception and decomposition of Wayside Status Messages (WSM).	
Action	reception
Semantic Group	Receptive
Recipient	obm
Theme	wayside status messages

A3: The Wayside Segment shall transmit information to the Automobile Segment in the form of Wayside Status Messages.	
Action	transmit
Semantic Group	Transmissive
Agent	wayside segment
Recipient	automobile segment
Theme	wayside status messages

that is easily comprehensible to a human stakeholder. The link rationale incorporates two types of knowledge: (1) relations that exist between domain concepts appearing across source and target artifacts, and (2) relations that exist between action frames.

4.3.1 Relations Between Domain Concepts

Relations between domain concepts are documented in the domain ontology. Link heuristics utilize thematic roles. For example, both action frames in Table 4.1 include thematic roles of *recipient* and *theme*. Furthermore, the action frames are matched using the *transmissive-receptive* heuristic. One of its rules states that if a recipient is present in both action frames, then the mapped dependencies (i.e. *OBM* and *automobile segment*) must exhibit an exact or hierarchical match in the domain ontology. The exact relationship between the concepts can be retrieved from the ontology and must then be included in the link rationales.

As DoCIT currently documents *equivalence*, *compositional*, and *hierarchical* relations between concepts, we define three corresponding patterns for generating rationales:

- **Equivalence Relations: "A and B are equivalent"**
  For example, the acronym expansion of WIU represents an equivalence relation and would generate the text: "WIU and Wayside Interface Unit are equivalent."
- **Compositional Relations "A is part of B"**
  For example, the match between *recipients* in *A2* and *A3* in Table 4.1 would generate the text: "OBM is part of Automobile Segment."
- **Generalization Relations: "A is a B"**
  For example, "driver" is a subclass of "train crew." If these two phrases appeared across source and target artifacts, then the link rationale would include the generated text "Driver is a train crew." From this example, we see the need for greater contextualization of the generated text so that the sentence makes more sense. In this case, a better sentence would clearly be "The driver is a member of the train crew." For now, we generate homogeneous text for each type of relation, which leads to somewhat awkward, albeit comprehensible, wording. We leave grammatical and other such improvements to our future work.

TABLE 4.2

TRACE LINK RATIONALE GENERATION EXAMPLES BY DOCIT

Source Artifact	*The DOH System shall include safety-critical integration of all authorities and indications of wayside and DC signal systems, and other similar digital I/O systems providing signal states, in order to provide warning and enforcement.*	**Target Artifact**	*The OBM shall treat DC signals as applying to the current allowable speed for the current lane occupied by the automobile.*
Source Action Frame	Action: include Semantic Group: Inclusive Agent: doh system Theme: dc signal system indication	Target Action Frame	Action: treat Semantic Group:Regulative Agent: obm Theme: dc signal
Heuristic	*Inclusive-Regulative*		
Rationale Pattern	"In order" + ("for" + *Agent2*) + "to regulate" + *Theme2* + *Agent1* + "must first incorporate" + *Theme1*		
Generated	OBM is part of DOH System. DC signal system indication and DC signal are equivalent. In order to regulate DC signals, the DOH System must first incorporate DC signal system indications.		

Source Artifact	*The DOHS shall predicatively enforce road bulletins, forms based authorities, road devices, and consist or lading speed restrictions, which includes automobile based permanent speed restrictions.*	**Target Artifact**	*The DOHS shall display and enforce all mandatory directives it holds at any given time.*
Source Action Frame	Action: enforce Semantic Group: Enforcive Agent: dohs Theme: road bulletin	Target Action Frame	Action: enforce Semantic Group: Enforcive Agent: dohs Theme: mandatory directive
Heuristic	*Basic (Groups must match)*		
Rationale Pattern	"Both artifacts involve" + *G(Agent1, Agent2*) + Action1 + "ing" + *G(Theme1, Theme2)*		
Generated	Both artifacts mention the concept of DOHS. Road Bulletin is part of Mandatory Directive. Both artifacts involve DOHS enforcing Mandatory Directive.		

Source Artifact	*The PCA pump shall detect air-in-line embolism.*	**Target Artifact**	*The upstream monitor measures drug flow into the pump and detects upstream occlusion.*
Source Action Frame	Action: detect Semantic Group: Calculative Agent: pca pump Theme: air-in-line embolism	Target Action Frame	Action: detect Semantic Group: Calculative Agent: upstream monitor Theme: upstream occlusion
Heuristic	*Basic (Groups must match)*		
Rationale Pattern	"Both artifacts involve" + *G(Agent1, Agent2*) + Action1 + "ing" + *G(Theme1, Theme2)*		
Generated	Upstream monitor is part of PCA pump. Air-in-line embolism is a upstream occlusion Both artifacts involve PCA pump detecting upstream occlusion.		

4.3.2 Relations Between Action Frames

Text explaining the relations between action frames involves organizing several matched pairs of thematic roles into a coherent sentence. It can be seen as the reverse process of generating the action frames from the original artifacts. We illustrate our approach using the earlier example of applying the *transmissive-receptive* heuristic to the artifacts shown in Table 4.1. The following pattern is used to generate the text:

"Both artifacts involve transmitting" + G(Theme1, Theme2) +
"to" + G(Recipient1, Recipient2).

RoleName1 and *RoleName2* indicate the thematic roles from source and target action frames respectively. The notation *G(RoleName1, RoleName2)* in the pattern represents the more general concept between the contents in *RoleName1* and *RoleName2* as defined in the domain ontology. If they exhibit a hierarchical relation, the general concept is the one closer to the root node. If they are compositionally related, the general concept is the one that has the other as its part. As a result, in our example in Table 4.1, this rationale pattern generates the text "Both artifacts involve transmitting Wayside Status Message to Automobile Segment" as a rationale for this trace link.

In Table 4.2, we show three examples of trace links established by DoCIT with the extracted action frames, the link heuristics applied, the rationale patterns, and finally the text generated and displayed to the user. The content in parentheses in the rationale pattern is optional to avoid redundancy.

DoCIT currently has 24 defined trace link heuristics, and it is likely that more will be added. Each of these heuristics requires its own rationale generation pattern to be defined. We have generated such patterns for 11 of the heuristics - as depicted in Table 4.3.

TABLE 4.3

ADDITIONAL ACTION-FRAME LINK RATIONALE GENERATION PATTERNS

Basic (Group must match)	"Both artifacts involve" + G(Agent1, Agent2) + Action1 + "ing" + G(Theme1, Theme2) "Both artifacts involve" + G(Theme1, Theme2) + Action1 + "ing" + G(Location1, Location2) + ("from" + G(InitialLocation1, InitialLocation2))
Motive-Motive	"Both artifacts involve" + G(Theme1, Theme2) + "entering" + G(Location1, Location2)
Motive-Permissive	"Both artifacts involve" + G(Theme,1 Theme2) + "entering" + Location1
Enforcive-Calculative	"In order" + ("for" + Agent1) + "to" + Acton1 + Theme1 + Agent2 + "must first" + Action2 + Theme2
Descriptive-Transmissive	Action2 + "ing" + Theme2 + "requires" + Action1 + Theme1
Enforcive-Enforcive	"Both artifacts involve" + G(Agent1, Agent2) + "enforcing" + G(Instrument1, Theme2)
Administrative-Inceptive	Action2 + "ing" + Theme2 + "requires" + Action1 + Theme1
Transmissive-Receptive	"Both artifacts involve transmitting" + G(Theme1, Theme2) + "to"+ G(Recipient1, Recipient2)
Permissive-Inclusive	"Both artifacts involve" + G(Agent1, Agent2) + "incorporating" + G(Theme1, Theme2)
Necessitative-Inspective	"In order" + ("for" + Agent1) + "to" + Acton1 + Theme1 + Agent2 + "must first" + Action2 + Theme2
Inclusive-Regulative	"In order" + ("for" + Agent2) + "to regulate" + Theme2 + Agent1 + "must first incorporate" + Theme1

4.4 Related Work

The closest related work is that of *Link Rationales* presented by Hull. Other traceability approaches that we are aware of which provide any explanation of generated trace links limit the description to a semantically meaningful label [134, 117, 105]. In the area of model driven development (MDD), much emphasis has been placed on automating trace link creation. Because of the inherently structured environment, link semantics are often well understood [98]. However, the link semantics also focus on the role of link types (i.e. all links between modules A and B) rather than explaining the specific reason for each individual link.

In the area of architectural design, researchers such as Burge et al. [21] and Kruchten [81] have used design rationales to document architectural design decisions, their justifications, alternatives, and trade-offs. Other similar approaches include the Architecture Design Decision Support System (ADDSS) [22], techniques for managing architectural knowledge (PAKME) [7], and Architecture Rationale and Element Linkage (AREL) [139]. However, these solutions focus on documenting rationales of the design itself, rather than describing the rationale for the trace link. While the link rationale might be inferable from the design rationale, it is certainly not the primary intent.

4.5 Conclusion

This chapter provides a preliminary introduction to a new area of work. Our approach identifies relations between domain concepts appearing across trace links and relations between action frames that capture complex domain knowledge in artifacts. The identified relations are shown to the user in the form of natural language text based on predefined rationale generation patterns. The rationales are designed to help project stakeholders understand and assess the trace links.

The quality of the generated rationales would mainly rely on three factors: the correct-

ness of the syntactic parser used in DoCIT, the completeness of the domain knowledge base, and richness of the rationale generation patterns. The first two factors also represent the primary limitations of the current DoCIT design. In the next chapter, therefore, we focus on how we utilize neural network to achieve semantically enhanced trace link generation without any dependence on syntactic parser and handcrafted domain knowledge.

CHAPTER 5

SEMANTICALLY ENHANCED SOFTWARE TRACEABILITY USING DEEP LEARNING TECHNIQUES

This chapter presents the work entitled "Semantically Enhanced Software Traceability Using Deep Learning Techniques" that was published in the *Proceeding of the 39th International Conference on Software Engineering* [61].

5.1 Introduction

In Chapter 3, we have proposed the *Domain-Contextualized Intelligent Traceability* (DoCIT) solution [58] as a proof of concept solution to investigate the integration of domain knowledge into the tracing process. We demonstrated that for a complex and safety-critical domain such as transportation vehicle communication and control domain, DoCIT returns accurate trace links achieving mean average precision (MAP) of 0.822 in comparison to 0.590 achieved using VSM. However, the cost of setting up DoCIT for a domain is non-trivial, as it requires carefully handcrafting a domain ontology, and manually defining trace link heuristics capable of reasoning over the semantics of the artifacts and associated domain knowledge. Furthermore, DoCIT depends upon a conventional syntactic parser to analyze the artifacts in order to extract meaningful concepts. The approach is therefore sensitive to errors in the parser, as well as terms missing from the ontology, and missing or inadequate heuristics. As such, DoCIT is effective but fragile, and would require significant effort to transfer into new project domains.

On the other hand, Deep Learning techniques have successfully been applied to solve many Natural Language Processing (NLP) tasks including parsing [133], sentiment analy-

sis [138], question answering [74], and machine translation [9]. Such techniques abstract problems into multiple layers of nonlinear processing nodes; they leverage either supervised or unsupervised learning techniques to automatically learn a representation of the language and then use this representation to perform complex NLP tasks. The goal of the work described in this chapter is to utilize deep learning to deliver a scalable, portable, and fully automated solution for bridging the semantic gap that currently inhibits the success of trace link creation algorithms. Our solution is designed to automate the capture of domain knowledge and the artifacts' textual semantics with the explicit goal of improving accuracy of the trace link generation task.

The approach we propose includes two primary phases. First, we learn a set of **word embeddings** for the domain using an unsupervised learning approach trained over a large set of domain documents. The approach generates high dimensional word vectors that capture distributional semantics and co-occurrence statistics for each word [114]. Second, we use an existing training set of validated trace links from the domain to train a Tracing Network to predict the likelihood of a trace link existing between two software artifacts. Within the tracing network, we adopt a **Recurrent Neural Network** architecture to learn the representation of artifact semantics. For each artifact (i.e. each regulation, requirement, or source code file etc.), each word is replaced by its associated vector representation learned in the word embedding training phase and then sequentially fed into the RNN. The final output of RNN is a vector that represents the semantic information of the artifact. The tracing network then compares the semantic vectors of two artifacts and outputs the probability that they are linked.

Given the need for an initial training set of trace links, our approach cannot be used in an entirely green field domain. However, based on requests from our industrial collaborators, we envision the following primary usage scenarios: (1) Train the tracing network on an initial set of manually constructed trace links for a project and then use it to automate the production of other links as the project proceeds, (2) Train the tracing network on the

complete set of trace links for a project and then use it to find additional links that may have been missed during the manual link construction process, and finally (3) Train the tracing network on the trace links for one project, or for specific types of artifacts in one project, and then apply it to other projects and artifact types within the same domain. In this chapter we focus on the first scenario.

We evaluate our approach on a large industrial dataset taken from the domain of Positive Train Control (PTC) systems to address two research questions:

RQ1: How should RNN be configured in order to generate the most accurate trace links?

RQ2: Is RNN able to significantly improve trace link accuracy in comparison to standard baseline techniques?

The remainder of the chapter is structured as follows. We first introduce deep learning techniques related to the tracing network in Section 5.2. The architecture of the tracing network is described in Section 5.3. Section 5.4 and 5.5 describe the process used to configure the tracing network, our experimental design, and the results achieved. Finally, in Section 5.7 to 5.8 we discuss threats to validity, related work, and conclusions.

5.2 Deep Learning for Natural Language Processing

Many modern deep learning models and associated training methods originated from research in artificial neural networks (ANN). Inspired by advances in neuroscience, ANNs were designed to approximate complex functions of the human brain by connecting a large number of simple computational units in a multi-layered structure. Based on ANNs, *Deep Learning* models feature more complex network connections in a larger number of layers. A benefit gained from the more complex structure is the ability to represent data features with multiple levels of abstraction; this is usually preferable to more traditional machine learning techniques, in which human expertise is essential to generate features of data for training. Back-propagation, proposed in [122], is widely recognized as an effective method for training deep neural networks; it indicates how the network should adapt its

internal parameters to better compute the representation in each layer. Before presenting our approach, we describe fundamental concepts of deep learning techniques, especially as related to NLP tasks. Furthermore, as our interest lies in comparatively evaluating different models for purposes of trace link creation, we describe these various techniques in some depth.

5.2.1 Word Embedding

Conventional NLP and information retrieval techniques treat unique words as atomic symbols and therefore do not take associations among words into account. To address this limitation, word embedding learns the representation of each word from a corpus as a continuous high dimensional vector such that similar words are close together in the vector space. In addition, the embedded word vectors encode syntactic and semantic relationships between words as linear relationships between word vector subtractions [101]. The use of learned word vectors is considered one of the primary reasons for the success of recent deep learning models for NLP tasks [40].

Among word embedding models, Skip-gram with negative sampling [101, 102] and GloVe [114] are the most popular models due to the notable improvement they bring to word analogy tasks over more traditional approaches such as Latent Semantic Analysis [101, 114]. Word embedding models are trained using unlabeled natural language text by utilizing co-occurrence statistics of words in the corpus. The Skip-gram model scans context windows across the entire training text to train prediction models [101]. Given the *center* word in the window of size T, the Skip-gram model maximizes the probability that the targeted word appears around the center word, while minimizing the probability that a random word appears around the center word. The GloVe model uses matrix factorization; however we do not discuss it further because it performs similarly to the Skip-gram with negative sampling approach but utilizes more system resources and is less robust [84].

5.2.2 Neural Network Structures

Deep learning solutions for NLP tasks utilize neural network techniques. *Feedforward* networks, also referred to as multi-layer perceptrons (MLPs), represent a traditional neural network structure [66]. Each layer of a MLP network is comprised of an affine transformation and non-linear point-wise activation function, expressed as $y = f(Wx + b)$, where x and y are respectively the input and output vectors for a layer and W and b are the weight matrix and bias vector for affine transformation. The number of layers defines the depth of the network. All layers except for the input and output layers are "hidden" and information flows from the input to the output without any feedback. The distinct values of W and b for affine transformation on each layer constitute the parameters of the neural network. Aimed at approximating complex functions that map input vectors to output vectors, MLPs lay the foundation for many other structures. However, the number of parameters in a fully connected MLP can grow extremely large as the width and depth of the network increases. To address this limitation, researchers have proposed various neural network structures targeting different types of practical problems. For example, convolutional neural networks (CNNs) are especially well suited for image recognition and video analysis tasks [83].

For NLP tasks, Recurrent neural networks (RNNs) are widely used and are recognized as particularly well suited to the unique needs of NLP [122]. In particular, RNN and its variants have produced significant breakthroughs in many NLP tasks including language modeling [100], machine translation [9], and semantic entailment [138]. In the following sections, we first introduce background information about RNN and then discuss several RNN variants that were explored and evaluated in this study.

5.2.3 Standard Recurrent Neural Networks (RNN)

RNNs are particularly well suited for processing sequential data such as text and audio. They connect computational units in a directed cycle such that at each time step t, a unit in the RNN not only takes input of the current step (i.e. the coming word embedding

from the text) but is also fed with the hidden state of the same unit from the previous time step $t-1$. This feedback mechanism simulates a "memory", so that a RNN's output is determined by current and prior inputs. Furthermore, because RNNs use the same unit (with the same parameters) across all time steps, they are able to process sequential data of arbitrary length. This is illustrated in Figure 5.1. At a given time step t with input vector x_t and its previous hidden output vector h_{t-1}, a standard RNN unit calculates its output as

$$h_t = tanh(Wx_t + Uh_{t-1} + b) \tag{5.1}$$

where W, U and b are the affine transformation parameters, and $tanh$ is the hyperbolic tangent function: $tanh(z) = (e^z - e^{-z})/(e^z + e^{-z})$.

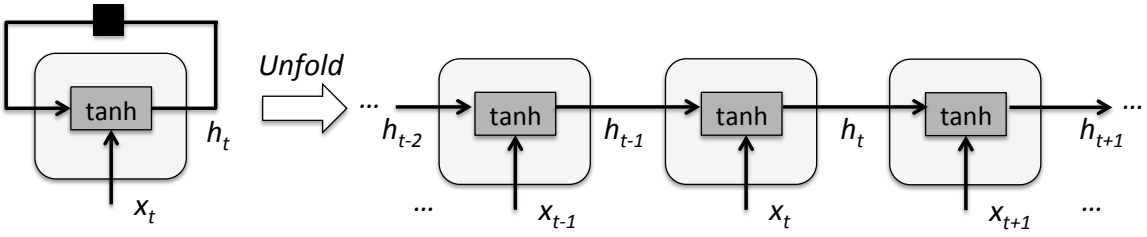


Figure 5.1. Standard RNN model (left) and its unfolded architecture through time (right). The black square in the left figure indicates a one time step delay.

A prominent drawback of the standard RNN model is that the network degrades when long dependencies exist in the sequence due to the phenomenon of exploding or vanishing gradients during back-propagation [12]. This makes a standard RNN model difficult to train. The exploding gradients problem can be effectively addressed by scaling down the gradient when its norm is bigger than a preset value (i.e. *Gradient Clipping*) [12]. To

address the vanishing gradients problem of the standard RNN model, researchers have proposed several variants with mechanics to preserve long-term dependency; these variants included Long Short Term Memory (LSTM) and the Gated Recurrent Unit (GRU).

5.2.4 Long Short Term Memory (LSTM)

LSTM networks include a memory cell vector in the recurrent unit to preserve long term dependency [70]. LSTM also introduces a gating mechanism to control when and how to read or write information to the memory cell. A gate in LSTM usually uses a sigmoid function $\sigma(z) = 1/(1 + e^{-z})$ and controls information throughput using a point-wise multiplication operation $\odot$. Specifically, when the sigmoid function outputs 0, the gate forbids any information from passing, while all information is allowed to pass when the sigmoid function output is 1. Each LSTM unit contains an *input* gate (i_t), a *forget* gate (f_t), and an *output* gate (o_t). The state of each gate is decided by x_t and h_{t-1} such that:

$$\begin{aligned} i_t &= \sigma(W^i x_t + U^i h_{t-1} + b^i) \\ f_t &= \sigma(W^f x_t + U^f h_{t-1} + b^f) \\ o_t &= \sigma(W^o x_t + U^o h_{t-1} + b^o) \end{aligned} \tag{5.2}$$

To update the information in the memory cell, a memory candidate vector $\tilde{c}_t$ is first calculated using the *tanh* function. This memory candidate passes through the input gate, which controls how much each dimension in the candidate vector should be "remembered". At the same time, the *forget* gate controls how much each dimension in the previous memory cell state c_{t-1} should be retained. The actual memory cell state c_t is then updated using the sum of these two parts.

$$\begin{aligned} \tilde{c}_t &= tanh(W^c x_t + U^c h_{t-1} + b^c) \\ c_t &= i_t \odot \tilde{c}_t + f_t \odot c_{t-1} \end{aligned} \tag{5.3}$$

Finally, the LSTM unit calculates its output h_t with an output gate as follows:

$$h_t = o_t \odot tanh(c_t) \tag{5.4}$$

Figure 5.2 (a) illustrates a typical LSTM unit. Using retained memory cell state and the gating mechanic, the LSTM unit "remembers" information until it is erased by the *forget* gate; as such, LSTM handles long-term dependencies more effectively. LSTM has been repeatedly applied to solve semantic relatedness tasks and has achieved convincing performance [138, 120]. These advances motivated us to adopt LSTM for reasoning semantics in the tracing task.

5.2.5 Gated Recurrent Unit (GRU)

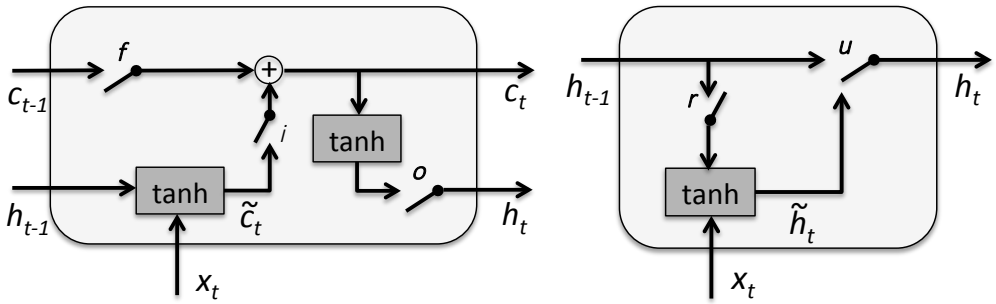
Finally, the recently proposed Gated Recurrent Unit (GRU) model also uses a gating mechanism to control the information flow within a unit; but it has a simplified unit structure and does not have a dedicated memory cell vector [25]. It contains only a *reset* gate r_t and an *update* gate u_t:

$$\begin{aligned} r_t &= \sigma(W^r x_t + U^r h_{t-1} + b^r) \\ u_t &= \sigma(W^u x_t + U^u h_{t-1} + b^u) \end{aligned} \tag{5.5}$$

In GRU networks, the previous hidden output h_{t-1} goes through the *reset* gate r_t and is sent back to the unit. An output candidate $\tilde{h}_t$ is then calculated using the gated h_{t-1} and the unit's current input x_t as follows:

$$\tilde{h}_t = \tanh(W^h x_t + U^h(r_t \odot h_{t-1}) + b^h) \tag{5.6}$$

The actual output of a unit h_t is a linear interpolation between the previous output h_{t-1} and the candidate output $\tilde{h}_t$, controlled by the *update* gate u_t. As such, the *update* gate



(a) LSTM (b) GRU

Figure 5.2. Comparison between single units from LSTM and GRU networks. In (5.2a), i, f and o are the input, forget and output gates respectively; c is the memory cell vector. In (5.2b), r is the reset gate and u is the update gate [26].

balances how much of the current output is updated using $\tilde{h}_t$ and h_{t-1}.

$$h_t = (1 - u_t) \odot h_{t-1} + u_t \odot \tilde{h}_t \tag{5.7}$$

Consequently, a GRU unit embeds long-term information directly into the hidden output vectors. Figure 5.2 compares the unit structures of LSTM and GRU networks. Desipte having a simpler structure, GRU has achieved competable results with LSTM on many NLP tasks [9, 26]. There's no decisive conclusion about which model is better. In this work, we compare the performance of LSTM and GRU and aim to find a more suitable model for the tracing problem.

5.2.6 Other RNN Variables

In addition to modifying the structure within a single RNN unit, the structure of the overall RNN network can vary. For instance, multi-layered RNNs [112] stack more than one RNN unit at each time step [127] with the aim of extracting more abstract information from the input sequence. In contrast, bi-directional RNNs [128] process sequential data in both forward and backward directions at the same time; this enables the output to be

influenced by both past and future data in the sequence. In this study, we explored the use of two-layered RNNs and bidirectional RNNs for generating trace links.

5.3 The Tracing Network

The goal of our study is to investigate the effectiveness of using various deep learning models and methods for generating accurate trace links. Based on our initial analysis of the strengths and weaknesses of current techniques, we decided to adopt word embeddings and RNN techniques. The task of trace link generation is essentially a textual comparison task in which the tracing network needs to leverage domain knowledge to understand the semantics of two individual artifacts and then to evaluate their semantic relatedness. Valid associations need to be established between related artifacts even when no common words are present. To achieve this goal, we first need to learn word embeddings from a domain corpus in order to effectively encode word relations. We also need to utilize such word embeddings in the tracing network structure to extract and compare their semantics. In this section, we describe how we design and train such tracing network.

5.3.1 Network Architecture

The design of the neural network architecture is shown in Figure 5.3. Given the textual content of a source artifact A_s and a target artifact A_t, each word in A_s and A_t is first mapped onto its vector representation through the *Word Embedding* layer. Such mappings are trained from the domain corpus using the Skip-gram model introduced in Section 5.2.1. The vectors of words in source artifact s_1, s_2, ..., s_m are then sent to the *RNN* layers sequentially and output as a single vector v_s representing its semantic information. In the case of the bidirectional-RNN, the word vectors are also sent in reverse order as s_m, s_{m-1}, ..., s_1. The target semantic vector v_t is generated in the same way using RNN layers. Finally, these two vectors are compared in the *Semantic Relation Evaluation* layers.

The *Semantic Relation Evaluation* layers in our tracing network adopt the structure

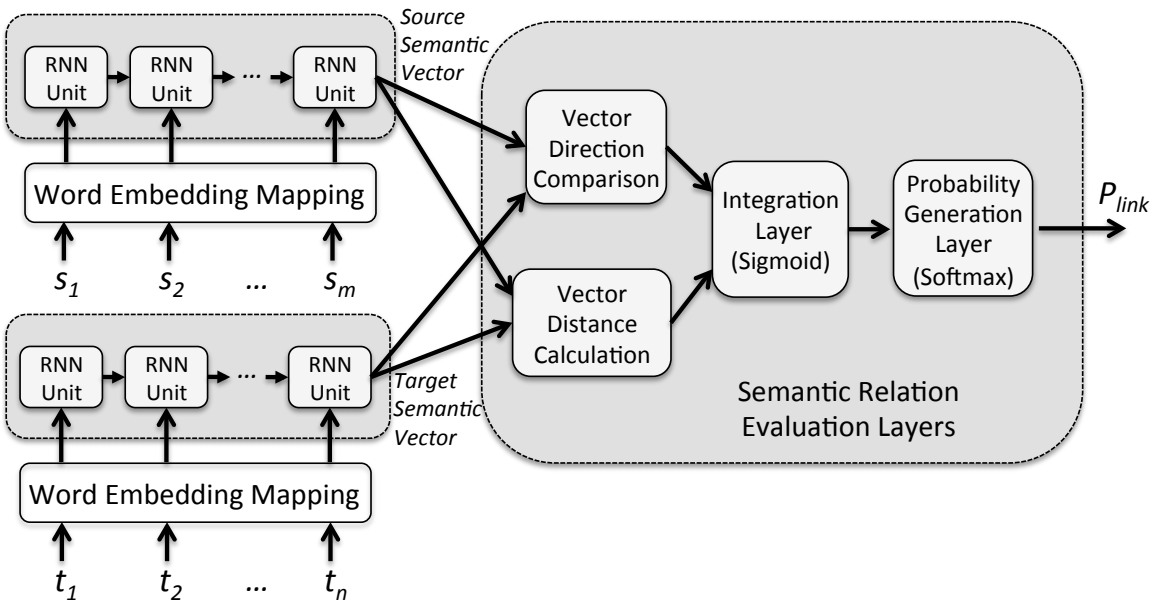


Figure 5.3. The architecture of the tracing network. The software artifacts are first mapped into embedded word vectors and go through RNN layers to generate the semantic vectors, which are then fed into the Semantic Relation Evaluation layers to predict the probability that they are linked.

proposed by Tai et al. [138], targeted to perform semantic entailment classification tasks for sentence pairs. The overall calculation of this part of the network can be represented as:

$$\begin{aligned} r_{pmul} &= v_s \odot v_t \\ r_{sub} &= |v_s - v_t| \\ r &= \sigma(W^r r_{pmul} + U^r r_{sub} + b^r) \\ p_{tracelink} &= softmax(W^p r + b^p) \end{aligned} \tag{5.8}$$

where

$$softmax(z)_j = e^{z_j} / \sum_{k=1}^{K} e^{z_k}, for j = 1, \ldots, K \tag{5.9}$$

Here, $\odot$ is the point-wise multiplication operator used to compare the *direction* of source and target vectors on each dimension. The absolute vector subtraction result, r_{sub},

represents the *distance* between the two vectors in each dimension. The network then uses a hidden *sigmoid layer* to integrate r_{pmul} and r_{sub} and output a single vector to represent their semantic similarity. Finally, the output *softmax layer* uses the result to produce the probability that a valid trace link exists; the result of a softmax function is a K-dimensional vector of real values in the range $(0, 1)$ that add up to 1 (K=2 in this case).

A concrete tracing network is built upon this architecture and is further configured by a set of network settings. Those settings specify the type of RNN unit (i.e. GRU or LSTM), the number of hidden dimensions in RNN units and the Semantic Relation Evaluation layers, and other RNN variables such as the number of RNN layers and whether to use bidirectional RNN. To address our first research question (RQ1) we explored several different configurations. We describe how we selected an optimized network settings in Section 5.4.2.

5.3.2 Training the Tracing Network

A powerful network is only useful when it can be properly trained using existing data and when it is generalizable to unseen data. To train the tracing network, we use the regularized negative log likelihood as our objective loss function to be minimized. This objective function is commonly used in categorical prediction models [10] and can be written as:

$$J(\theta) = -\frac{1}{N}\sum_{i=1}^{N} logP(Y = y^i|x^i, \theta) + \frac{\lambda}{2}||\theta||_2^2 \tag{5.10}$$

where θ indicates the network parameters that need to be trained, N is the total number of examples in the training data, x^i is the input of the *ith* training example, y^i is the actual category of that example (i.e. link or non-link); as a result, $P(Y = y^i|x^i, \theta)$ represents the network's prediction on the correct category given the current input and parameters. The second part of the loss function represents a L^2 parameter regularization that prevents

overfitting, where $\|\theta\|_2$ is the Euclidean norm of θ and λ controls the strength of the regularization.

Based on this loss function, we used a stochastic gradient descent method [140] to update the network parameters. According to this method, a typical training process is comprised of a number of epochs. Each *epoch* iterates through all of the training data one time to train the network; as such, the overall training process uses all the training data several times until the objective loss is sufficiently small or fails to decrease further. In each epoch, the training data is further randomly divided into a number of "mini batches;" each contains one or more training datapoints. After each batch is processed, a gradient of parameters is calculated based on the loss function. The network then updates its parameters based on this gradient and a "learning rate" that specifies how fast the parameters move along the gradient direction.

During training, we adopted an adaptive learning rate procedure [140] to adjust the learning rate based on the current training performance. To help the network converge, we also decreased the learning rate after each epoch until epoch τ, such that the learning rate at epoch k is determined by:

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau \tag{5.11}$$

where ϵ_0 is initial learning rate, $\alpha = k/\tau$. In our experiment, ϵ_τ is set to $\epsilon_0/100$, and τ set to 500.

Based on these general methods, a tracing network training process is further determined by a set of predefined *hyper-parameters* that help steer the learning behavior. Common hyper-parameters include learning rate, gradient clip value, regulation strength (λ), the number of datapoints in a mini batch (i.e. mini batch size), and the number of epochs included in the training process. The techniques for selecting hyper-parameters are described in Section 5.4.2.

5.4 Experiment Setup

In this section, we explain the methods used to (1) prepare data, (2) systematically tune the configuration (i.e. network settings and hyper-parameters) of the tracing network, and (3) compare the performance of the best configuration against other popular trace evaluation methods.

5.4.1 Data Preparation

To train the word embeddings, we used a corpus from the PTC domain that is comprised of 52.7MB of clean text extracted from related domain documents and software artifacts. The original corpus of domain documents was collected from the Internet by our industrial collaborators as part of their initial domain analysis process. We also added the latest Wikipedia dump containing about 19.92GB of clean text to the corpus and used it for one variant of the word embedding configuration. All documents were preprocessed by transforming characters to lower-case and removing all non-alpha-numeric characters except for underscores and hyphens.

To train and evaluate other parts of the tracing network, we used PTC project data provided by our industrial collaborators. The dataset contains 1,651 Software Subsystem Requirements (SSRS) as source artifacts and 466 Software Subsystem Design Descriptions (SSDD) as target artifacts. Each source artifact contained an average of 33 tokens and described a functional requirement of the Back Office Server (BOS) subsystem. Each target artifact contained an average of 99 tokens and specified design details. There were 1,387 trace links between SSRS and SSDD artifacts, all of which were constructed and validated by our industrial collaborators. This dataset is considerably larger than those used in most previous studies on requirements traceability [6, 92, 65] and the task of creating links across such a large dataset represents a challenging industrial-strength tracing problem. We randomly selected 45% of the 769,366 artifact pairs from the PTC project dataset (i.e.

1,651 $\times$ 466) for inclusion in a training set, 10% for a development set, and 45% as a testing set. Given a fixed tracing network configuration, the training set was used to update the network parameters (i.e. the weight and bias for affine transformation in each layer) in order to minimize the objective loss function. The development set was used to select the best general model during an initial training process to ensure that the model was not overtrained. The test data was set aside and only used for evaluating the performance of the final network model.

Software project data exhibits special characteristics that impact the training of a neural network [60]. In particular, the number of actual trace links is usually very small for a given set of source and target artifacts compared to the total number of artifact pairs. In our dataset, among all 769,366 artifact pairs, only 0.18% are valid links. Training a neural network using such an unbalanced dataset is very challenging. A common and relatively simple approach for handling unbalanced datasets is to weight the minority class (i.e. the linked artifacts) higher than the majority class (i.e. the non-linked ones). However, in the gradient descent method, a larger loss weighting could improperly amplify the gradient update for the minority class making the training unstable and causing failure to converge. Another common way to handle unbalanced data is to downsample the majority case in order to produce a fixed and balanced training set. Based on initial experimentation we found that this approach did not yield good results because the examples of non-links used for training the network tended to randomly exclude artifact pairs that lay at the frontier at which links and non-links are differentiated. Furthermore, based on initial experimentation, we also ruled out the upsampling method because this considerably increased the size of the training set, excessively prolonging the training time.

Based on further experimentation we adopted a strategy that dynamically constructed balanced training sets using sub-datasets. In each epoch, a balanced training set was constructed by including all valid links from the original training set as well as a randomly selected equal number of non-links form the training set. The selection of non-links was

updated at the start of each epoch. This approach ensured that over time the sampled non-links used for training were representative and preserved an equal contribution of links and non-links during each epoch. Our initial experimental results showed this technique to be effective for training our tracing network.

5.4.2 Model Selection and Hyper-Parameters Optimization

Finding suitable network settings and a good set of hyper-parameters is crucial to the success of applying deep learning methods to practical problems [115]. However, given the running time required for training, the search space of all the possible combinations of different configurations was too large to provide full coverage. We therefore first identified several configurations that were expected to produce good performance. This was accomplished by manually observing how training loss changed during early epochs and following heuristics suggested in [11]. We then created a network configuration search space centered around these manually identified configurations; our search space is summarized in table 5.1. We conducted a *grid search* and trained all the combinations of each configuration in Table 5.1 using the training set and then compared their performance on the development set to find the best configuration. We describe our search space below.

For learning the word embeddings, we used the Skip-gram model provided by the Word2vec tool [101]. We trained the word vectors with two settings: 50-dimension vectors using the PTC corpus only and 300-dimension using both PTC and Wikipedia dump. The number of dimensions are set differently because the PTC corpus (38,771 tokens) contains considerably less tokens than the PTC + Wikipedia dump (8,025,288 tokens). While a smaller vector dimension would result in faster training, a larger dimension is needed to effectively represent the semantics of all tokens in the latter corpus.

To compare which variation of RNN best suits the tracing network, we evaluated GRU, LSTM, bi-directional GRU (BI-GRU), bi-directional LSTM (BI-LSTM) with both 1 and 2

TABLE 5.1

TRACING NETWORK CONFIGURATION SEARCH SPACE

Word Embedding Source	PTC docs – 50 dim, PTC docs + Wikipedia dump – 300 dim
RNN Unit Type	GRU, LSTM, BI-GRU, BI-LSTM, (AveVect as baseline)
RNN Layer	1, 2
Hidden Dimension	RNN30 + Intg10, RNN60 + Intg20
Init Learning Rate (lr)	1e-03, 1e-02, 1e-01
Gradient Clip Value (gc)	10, 100
Regularization Strength λ	1e-04, 1e-03
Mini Batch Size	1
Epoch	60

layers introduced in Section 5.2.3. The hidden dimensions in each RNN unit were set to either 30 or 60, while the hidden dimensions for the Integration layer were set to 10 or 20 correspondingly. As a baseline method, we also replaced the RNN layers with a bag-of-word method in which the semantic vector of an artifact is simply set to be the average of all word vectors contained in the artifact ("AveVect" in Table 5.1). We also summarize the search space for other hyper-parameters of the tracing network in Table 5.1.

5.4.3 Comparison of Tracing Methods

In practical requirements tracing settings, a tracing method returns a list of candidate links between a source artifact, serving the role of user query, and a set of target artifacts. An effective algorithm would return all valid links close to the top of the list. The effectiveness of a tracing algorithm is therefore often measured using Mean Average Precision (MAP). The definition of MAP was introduced in Section 2.2.3

We computed MAP using the test dataset only, and compared the performance of our tracing network with other popular tracing methods, i.e. Vector Space Model (VSM) and Latent Semantic Indexing (LSI). To make a fair comparison, we also optimized the configurations for the VSM and the LSI methods using a Genetic Algorithm to search through an extensive configuration space of preprocessors and parameters [89]. Finally, we configured VSM to use a local Inverse Document Frequency (IDF) weighting scheme when calculating the cosine similarity [65]. LSI was reduced to 75% dimensions. For both VSM and LSI we preprocessed the text to remove non alpha-numeric characters, remove stop words, and to stem each word using Porter's stemming algorithm.

We also evaluated the results by plotting a precision vs. recall curve. The graph depicts *recall* and *precision* scores at different similarity or probability values. The Precision-Recall Curve thus shows trade-offs between precision and recall and provides insights into where each method performs best – for example, whether a technique improves precision at higher or lower levels of recall [20]. A curve that is farther away from the origin indicates

better performance.

5.5 Results and Discussion

In this section, we report (1) the best configurations found in our network configuration search space, (2) the performance of the tracing network with the best configuration compared against VSM and LSI, and (3) the performance of the tracing network when trained with a larger training set of data.

5.5.1 What Is the Best Configuration for the Tracing Network?

This experiment aims to address the first research question (RQ1). When optimizing the network configuration of the tracing network, we first selected the best configuration for each RNN unit type; Table 5.2 summarizes these results. We found that the best configurations for all four RNN unit types were very similar: one layer RNN model with 30 hidden dimensions and an Integration layer of 10 hidden dimensions, learning rate of 1e-02, gradient clip value of 10, and λ of 1e-04. Performance varies for different RNN unit types.

Figure 5.4 illustrates the learning curves on the training dataset of the four configurations. All four RNN unit types outperformed the Average Vector method. This supports our hypothesis that word order plays an important role when comparing the semantics of sentences. Both GRU and BI-GRU achieved faster convergence and more desirable (smaller) loss than LSTM and BI-LSTM. Although quite similar, the bidirectional models performed slightly better than the unidirectional models for both GRU and LSTM on the training set; the bidirectional models also achieved better results on the development dataset compared to their unidirectional counterparts. As a result, the overall best performance was achieved using **BI-GRU** configured as shown in Table 5.2.

We also found that in three of the four best configurations, the word embedding vectors

TABLE 5.2

BEST CONFIGURATION FOR EACH RNN UNIT TYPE

RNN Unit	Dev. Loss	Word Emb.	L	D_r	D_s	lr	gc	λ
BI-GRU	0.1045	PTC	1	30	10	0.01	10	0.0001
GRU	0.1301	PTC	1	30	10	0.01	10	0.0001
BI-LSTM	0.1434	PTC	1	30	10	0.01	100	0.0001
LSTM	0.2041	PTC+Wiki	1	30	10	0.01	10	0.0001

L – number of layers in the RNN model, D_r – hidden dimension in RNN unit, D_s – hidden dimension in Integration layer, lr – initial learning rate, gc – gradient clipping value, λ – regularization strength

were trained using the PTC corpus alone. We speculate that one reason the PTC-trained word vectors performed better than the PTC+Wiki-trained vectors is due to differences in content of the two corpora. The PTC+Wiki corpus contains significantly more words that are used in diverse contexts because the majority of articles in the Wiki corpus are not related to the PTC domain. In the case of words that appear commonly in Wiki articles but convey specific meanings in the PTC domain (e.g. message, administrator, field, etc.), their context in more general articles is likely to negatively affect the reasoning task on domain specific semantics when there is insufficient training data to disambiguate their usage. We also tested the tracing network performance using 300-dimension word embedding trained by the PTC corpus. The result is similar to the best configuration found in Table 5.2. These findings suggest that using only the domain corpus to train word vectors with a reasonable size is more computationally economical and can yield better results.

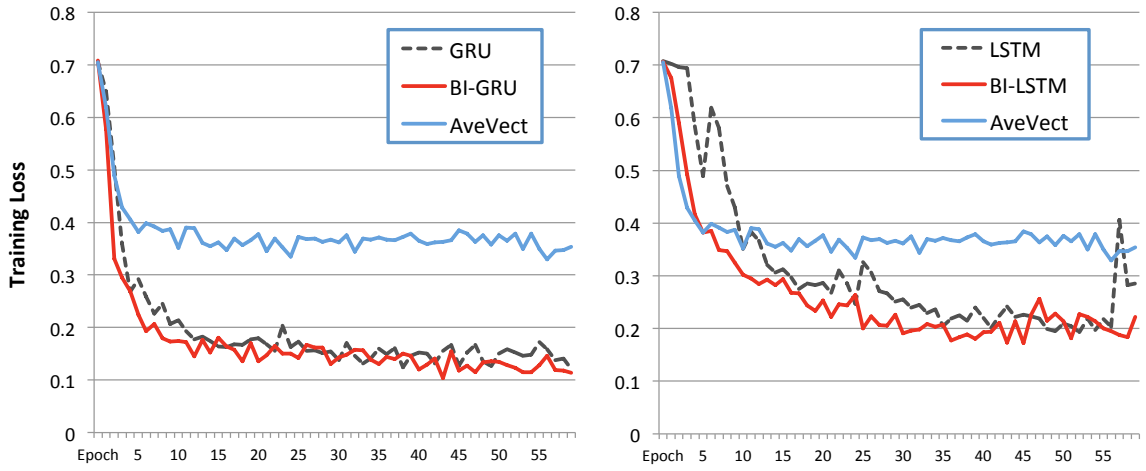


Figure 5.4. Comparison of learning curves for RNN variants using their best configurations. GRU and BI-GRU (left) converged faster and achieved smaller loss than LSTM and BI-LSTM (right). They all outperformed the baseline.

5.5.2 Does the Tracing Network Outperform Leading Trace Retrieval Algorithms?

We now evaluate whether the best configuration of the tracing network (i.e. BI-GRU with configuration shown in Table 5.2) outperforms leading trace retrieval algorithms. Links were therefore generated for each source and target artifact pair in the test set (45% total data) and for each source artifact ranked by descending probability scores. Average Precision (*AP*) was calculated using Equation 2.4. We then compared the APs for our tracing network against those generated using the best performing VSM and LSI configurations. Our tracing network was able to achieve a **MAP** of 0.598; this value is 41% higher than that achieved using VSM (**MAP** = 0.423) and 32% higher than LSI (**MAP** = 0.451). We conducted a Friedman test and found a statistically significant difference among the AP values associated with the three methods ($\chi^2(2) = 89.40$, $p < .001$). We then conducted three pairwise Wilcoxon signed ranks tests with Bonferroni p-value adjustments. Results indicated that the APs associated with our tracing network were significantly higher ($M = 0.598$, $SD = 0.370$) than those achieved using VSM ($M = 0.423$, $SD = 0.391$; $p < .001$) and LSI ($M = 0.451$, $SD = 0.400$; $p < .001$); in contrast, there was no significant difference

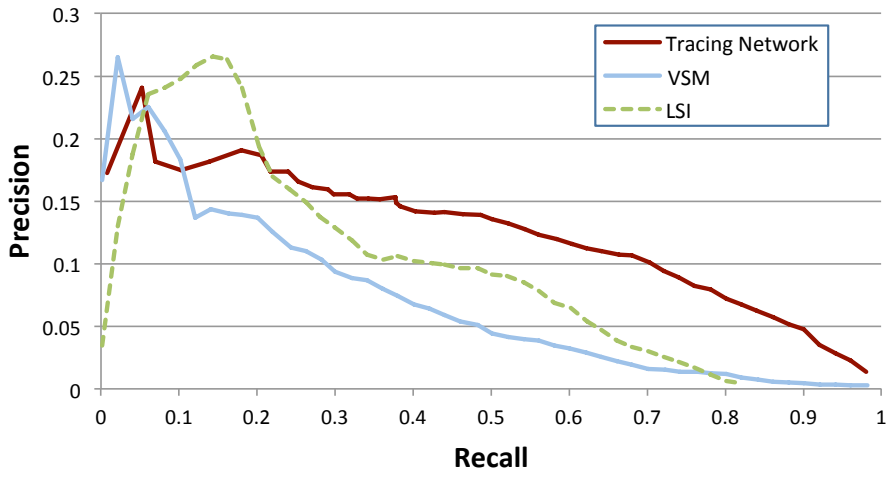
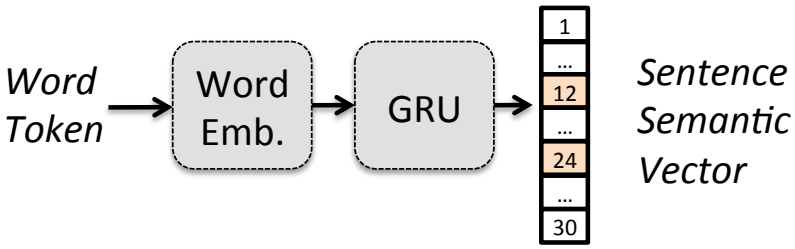


Figure 5.5. Precision-Recall Curve on test set – 45% total data

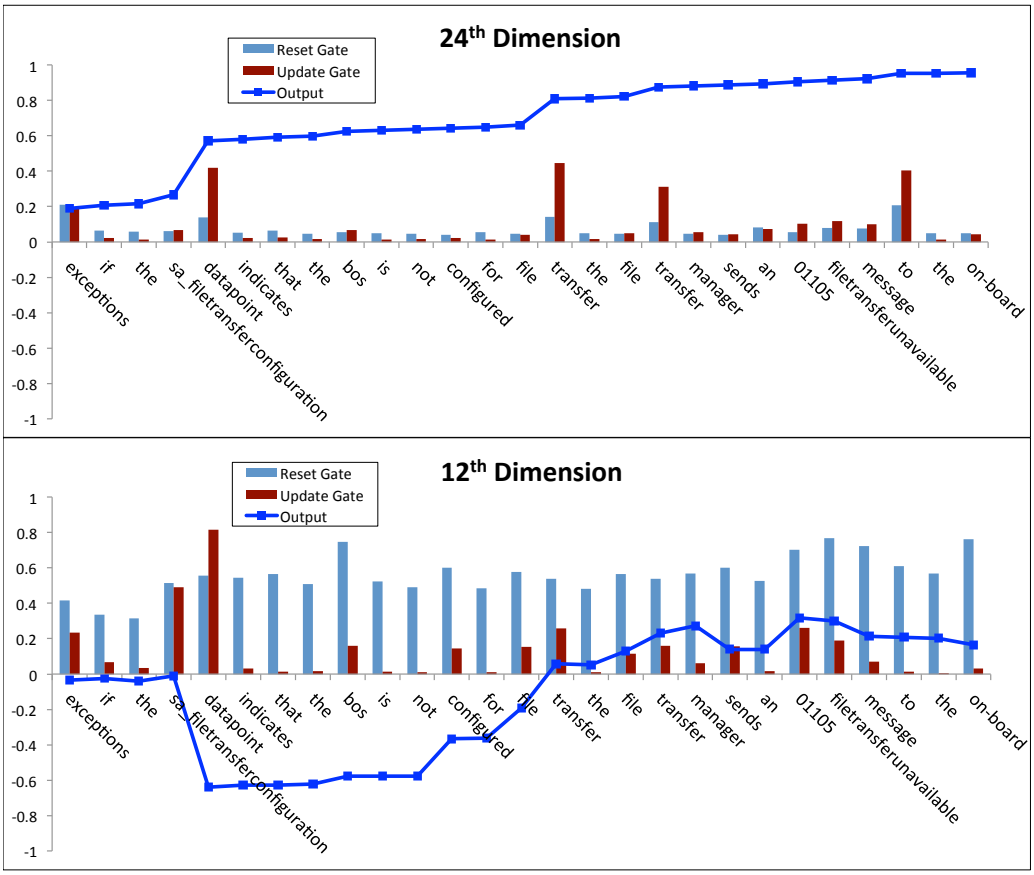
when comparing APs for VSM versus those for LSI.

When comparing the Precision-Recall Curves for the three methods (Figure 5.5), we observed that the tracing network outperformed VSM and LSI at higher levels of recall. Given the goal to achieve close to 100% recall when performing tracing tasks, this is an important achievement. Precision improved notably when recall was above 0.2. This improvement can be attributed to the fact that the tracing network is able to extract semantic information from artifacts and to reason over associations between them.

While it is almost impossible to completely decipher how GRU extracts a semantic vector from natural language [76], observing gate behavior when GRU processes a sentence can provide some insights into how GRU performs this task so well. As an example, we examine the gate behavior when our best performing GRU processes the following artifact text: "Exceptions: If the SA_FileTransferConfiguration datapoint indicates that the BOS is not configured for file transfer, the File Transfer Manager sends an 01105_FileTransferUnavailable message to the On-board." The gate behavior varies among dimensions in the sentence semantic vector. In Figure 5.6, we show the levels of the reset and the update gates and the change of the output values on two of the 30 dimensions after



(a) GRU output at one time step



(b) GRU gate behavior

Figure 5.6. The reset gate and update gate behavior in GRU and their corresponding output on the 24th and 12th dimension of sentence semantic vector. The x-axis indicates the sequential input of words while the y-axis is the value of reset gate, update gate and output after taking each word.

GRU takes each word from this artifact. Unlike LSTM, the GRU does not have an explicit memory cell; however, the unit output itself can be considered as a *persisting memory*, whereas the output candidate in Equation 5.6 acts as a *temporary memory*. Recall that the **reset gate** controls how much previous output adds to the current input when determining the output candidate (i.e. temporary memory); in contrast, the **update gate** balances the contributions of this temporary memory and the previous output (i.e. persisting memory) to the actual current output. From Figure 5.6b, we observe that for the 24th dimension, the reset gate is constantly small. This means that the temporary memory is mostly decided by the current input word. The update gate for this dimension was also small for most of the input words until the words *datapoint*, *transfer* and *to* came. Those spikes indicate moments when the temporary memory was integrated into the persisting memory. As such, we can speculate that this semantic vector dimension functions to accumulate information from specific keywords across the entire sentence. Conversely, for the 12th dimension, the reset gate was constantly high, indicating that the information stored in the temporary memory was based on both previous output and current input. Therefore, the actual output is more sensitive to local context rather than a single keyword. This is confirmed by the fluctuating shape of the actual output shown in the figure. For example, the output value remained in the same range until the topic was changed from "datapoint indication" to "message transfer". We believe that this versatile behavior of the gating mechanism might enable GRU to encode complex semantic information from a sentence into a vector to support trace link generation and other challenging NLP tasks.

From Figure 5.5, we also notice that the tracing network hits a glass ceiling for improving precision above 0.27. We consider this to be caused by its inability to rule out some false positive links that contain valid associations. For example, the tracing network assigns a 97.27% probability of a valid link between artifact "The BOS administrative toolset shall allow an authorized administrator to view internal errors generated by the BOS" and the artifact "The MessagesEvents panel provides the functionality to view message and event

logs. The panel provides searching and filtering capabilities where the user can search by a number of parameters depending on the type of data the user wants to view." There are direct associations between these artifacts: MessagesEvents panel is part of the BOS administrative toolset for viewing message and event log, administrator is a system user and internal errors generated by the BOS is an event. But this is not a valid link because MessagesEvents panel only displays messages and events related to external Railroad Systems rather than to internal events. It is likely that the tracing network fails to exclude this link because it has not been exposed to sufficient similar negative examples in the training data. As we described in Section 5.4.1, every positive example is used while negative examples (i.e. non-links) are randomly selected during each epoch. We plan to explore more adequate methods for handling unbalanced data problems caused by characteristics of the tracing data in our future work.

5.5.3 How Does the Tracing Network React to More Training Data?

The number of trace links tends to increase as a software project evolves. To explore the potential impact of folding them into the training set, we increased the training dataset to 80%. We randomly selected part of the test data and moved it into the training set to reach 80%, while retaining the remaining data in the testing set. Using the same configuration as described in Section 5.4.2, we then retrained the tracing network. Because the size of the test set decreased to 10%, we could not make direct comparisons to our previous results. Instead we used both of the trained tracing networks (i.e. trained with 45% and 80% of the data respectively) to generate trace links against the same small test set (sized at 10%). To reduce the effect of random data selection, we repeated this process five times and report the average results.

With a larger training set, the *MAP* was 0.834 compared to 0.803 for the smaller training set. Results are depicted in the Precision-Recall Curve in Figure 5.7. With increased training data, the network can better differentiate links and non-links, and therefore im-

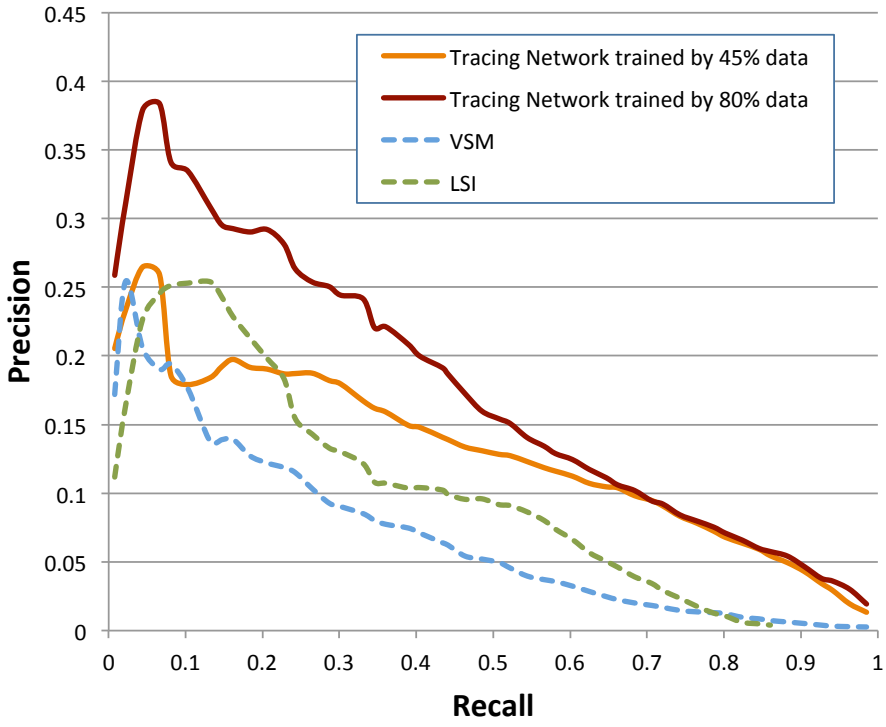


Figure 5.7. Precision-Recall Curve on test set – 10% total data.

prove both precision and recall. Improvements were observed especially at low levels of recall.

The performance of the tracing network trained using 80% of data was again compared against VSM and LSI for this larger training set. A Friedman test identified a statistically significant difference among the APs associated with the three methods on this new dataset division ($\mathcal{X}^2(2) = 141.11$, $p < .001$). Using pairwise Wilcoxon signed ranks tests with Bonferroni p-value adjustments, we found that our tracing network performed significantly better (**MAP** = 0.834) than VSM (**MAP** = 0.625; $p < .001$) and LSI (**MAP** = 0.637; $p < .001$). As such, we address RQ2 and conclude that in general our tracing network improved trace link accuracy in comparison to both VSM and LSI, and that improvements were more marked as the size of the training set increased. We expect additional improvements by reconfiguring the tracing network for use with a larger training set [11]. However, we also

observed that when using the original training set (i.e. 45% of data), our tracing network only outperformed LSI at higher levels of recall as shown in Figure 5.5 and Figure 5.7. In future work, we plan to explore the trade-offs between these two methods for specific data features, and to further improve the performance of the tracing network.

5.6 Threats to Validity

Two primary threats to validity potentially impact our work. First, due to the challenge of obtaining large industrial datasets including artifacts and trace links, and the time needed to experiment with different algorithms for learning word embeddings and generating trace links, our work focused on a single domain of Positive Train Control. As a result, we cannot claim generalizability. However, the PTC dataset included text taken from external regulations, and written by multiple requirements engineers, systems engineers, and developers. The threat to validity arises primarily from the possibility that characteristics of our specific dataset may have impacted results of our experiment. For example, the size of the overall dataset, the characteristics of the vocabulary used, and/or the nature of each individual artifact, may make our approach more or less effective. In the next phase of our work we will evaluate our approach on additional datasets.

Second, we cannot guarantee that the trace matrix used for evaluation is 100% correct. However, it was provided by our industrial collaborators and used throughout the project. The metrics we used (i.e. MAP, Recall, and Precision) are all accepted research standards for evaluating trace results [65]. To avoid comparison against a weak baseline, we compared our approach against standard baselines of VSM and LSI; both were configured by using a Genetic Algorithm to search for the best configuration options.

5.7 Related Work

Most of the related work on traceability techniques has been introduced and discussed in earlier chapters. The majority of those work focus on the term-matching based methods. While researchers have proposed techniques that more closely mimic the way human analysts reason about trace links and perform tracing tasks [57, 95], there is very limited work in this area. Our prior work with DoCIT, described in Chapter 3, is one exception [57]. DoCIT utilizes both ontology and heuristics to reason over concepts in the domain in order to deliver accurate trace links. However, as previously explained, DoCIT requires non-trivial setup costs to build a customized ontology and heuristics for each domain, and is sensitive to flaws in syntactic parsing. In contrast, the RNN approach described in this paper requires only a corpus of domain documents and a training set of validated trace links.

On the other hand, deep learning has been successfully applied to many software engineering tasks. For example, Lam et al. combined deep neural network with information retrieval techniques to identify buggy files in bug reports [82]. Wang et al. utilized a deep belief network to extract semantic features from source code for the purpose of defect prediciton [142]. Raychev et al. adopted RNN and N-gram to build the language model for the task of synthesizing code completions [119]. We were not able to find work that applied deep learning techiniques to traceability tasks in our literature review.

5.8 Conclusions

In this chapter, we have proposed a neural network architecture that utilizes word embedding and RNN techniques to automatically generate trace links. The Bidirectional Recurrent Gated Unit effectively constructed semantic associations between artifacts, and delivered significantly higher MAP scores than either VSM or LSI when evaluated on our large industrial dataset. It also notably increased both precision and recall. Given an initial

training set of trace links, our tracing network is fully automated and highly scalable. In future work, we will focus on improving precision of the tracing network by identifying and including more representative negative examples in the training set.

The tracing network is currently trained to process natural language text; however, we will investigate techniques for applying it to other types of artifacts such as source code or formatted data. Finally, given the difficulty and limitations of acquiring large corpora of data we will investigate hybrid approaches that combine the neural network with human knowledge or other automated tracing techniques. In summary, the findings we have presented in this chapter have demonstrated that deep learning techniques can be effectively applied to the tracing process. We see this as a non-trivial advance in our goal of automating the creation of accurate trace links in industrial-strength datasets.

CHAPTER 6

TOWARDS MORE GENERALIZABLE TRACING SOLUTIONS

6.1 Introduction

The tracing network proposed in Chapter 5 has demonstrated that deep learning based tracing solutions can be applied to solve software traceability problems. It utilizes a collection of domain documents to learn word representations, leverages a relatively large number of trace links to train the neural network parameters, and has achieved notable improvements on both precision and recall for a large industrial dataset. However, the tracing network faces unique challenges that may affect its generalizability when compared to similar solutions applied to tasks in the general Natural Language Processing domain. For many general NLP tasks, large training datasets can be created by web-mining or crowdsourcing with a reasonable amount of effort. This makes it practical to train complex deep learning models with a great number of parameters. On the contrary, it is significantly more difficult to create large sets of trace links for software engineering projects as this requires specific domain knowledge and software development expertise. As a result, the majority of current traceability datasets are small, especially those available for academic research. Furthermore, industrial datasets in different projects or domains often exhibit distinct characteristics. For example, large projects in complex domains often produce a vast number of artifacts. Many of those artifacts, for example their requirements or associated regulations, are often long and complicated and understanding them requires significant domain knowledge. On the other hand, artifacts from less technical domains are normally shorter and more straightforward. Deep learning methods therefore need to be applied to datasets with different sizes and features in order to increase their practical value.

Another important question is how to effectively make use of pre-trained deep learning models across different software engineering projects and domains. For general NLP tasks, it is not uncommon for models trained on one dataset to be reused or retrained for other tasks to boost their performance [77, 35]. Nevertheless, it is less common for software engineering tasks, especially for those that involve processing complicated artifact types such as regulations, software requirements, and design specifications. This may be explained by the fact that the terms used in those artifacts are often domain or context specific, making the trained models less adaptable.

As an extension of the tracing network and an initial exploration towards more generalizable tracing solutions, in this chapter, we aim to investigate: (1) whether the performance of the tracing network proposed in Chapter 5 can be further improved with existing training data; and (2) whether the tracing network can be applied to software traceability datasets with different characteristics. As a future direction, we also discuss potential benefits and challenges for the tracing network to use pre-training techniques in order to learn sentence representations.

6.2 Negative Link Sampling

As introduced in Section 5.4.1, we construct the dataset that is used for training and evaluating the tracing network by pairing every source artifact with every target artifact from the collection of software artifacts. If this pair represents a valid trace link between source and target artifacts (i.e. a user created link), then this data point is treated as a positive example in our dataset. Otherwise, the data point represents a negative example. Let us use L_{pos} to denote the set containing all the positive links from this dataset and L_{neg} for the set containing all the negative links. Therefore, the entire dataset is $L_{pos} \cup L_{neg}$. For training the tracing network, the ideal objective function would be:

$$\underset{\theta}{\operatorname{argmax}} \prod_{x \in L_{pos}} P(Y = 1|x, \theta) \prod_{x \in L_{neg}} P(Y = 0|x, \theta) \tag{6.1}$$

This means that the tracing network should be trained to maximize the probability of assigning 1 to all positive links and 0 to all negative links. In Section 5.4.1, we also described one of the unique characteristics of our dataset that it is highly unbalanced; i.e. the majority of the data points represent negative links. The dataset used in Chapter 5 contains $1,651 \times 466$ data points of which 99.982% are negative links. This characteristic of the tracing dataset poses a problem to the tracing network such that if we use all the data points equally the negative links will dominate the tracing process and consequently produce a tracing network that assigns values close to 0 to all data points. Moreover, it would be extremely time consuming to train the network with such a large number of data points. To address these issues, we therefore downsample the negative links during the actual training process. In other words, we create a subset of L_{neg} that are only k times as large as L_{pos} to approximate the objective 6.1. We use $\tilde{L_{neg}}$ to denote this downsampled negative links set.

In the experiment conducted in Section 5.4, we randomly split all the data points into three parts, representing the training set, development set and testing set. We chose to balance the size of L_{pos} and $\tilde{L_{neg}}$ of the training set during the training process (i.e. $k = 1$). However, such a balancing mechanism may result in unrepresentative negative links selections. In this section, we explore how the tracing network would behave when selecting different values of k.

6.2.1 Experiment Setup

To investigate the effect of negative link sampling on the performance of the tracing network, we use the same dataset from the Positive Train Control (PTC) domain described

in Section 5.4.1. We use 80% of the dataset for training, 10% for selecting the hyper-parameters of the tracing network, and the remaining 10% for evaluation.

We fix the RNN unit type as the Bidirectional Gated Recurrent Unit (BI-GRU) in this experiment and all other experiments reported in this chapter because BI-GRU was identified as the best model for the tracing network in our earlier work. We introduce the parameter k representing the negative links sampling ratio during the configuration search for the best hyper-parameters. The value of k is set to be 1, 5, or 10. We then find the best configuration of the tracing network for each value of k following the same procedure described in Section 5.5.1. We use the best configuration to train the tracing network and finally report its performance on the testing dataset. For comparison, we also report the results obtained using VSM and LSI tracing methods that are also optimized for this dataset.

6.2.2 Global Average Precision Measure

In Section 5.1, we have envisioned three usage scenarios of the tracing network to: (1) automate the generation of new links based on partially constructed trace links, (2) find missing trace links within a complete set of existing links, and (3) establish traceability in a new project based on trace links created for another project in the same domain. There are cases where instead of inspecting the returned list for each source artifact separately, the user would expect the tracing methods to return all the potentially valid links across the whole dataset, such as the second scenario described above. The MAP measure cannot properly reflect how the tracing methods could contribute to such scenario since it only measures the relative ranks of the target artifacts for individual sources. We therefore introduce another metric, **Global Average Precision (Global AP)** to evaluate the tracing methods. Global AP is calculated by first ranking all the source and target artifact pairs as one single list according to their similarity scores produced by the tracing method, and then computing the Average Precision for this list using the Formula 2.4 introduced in

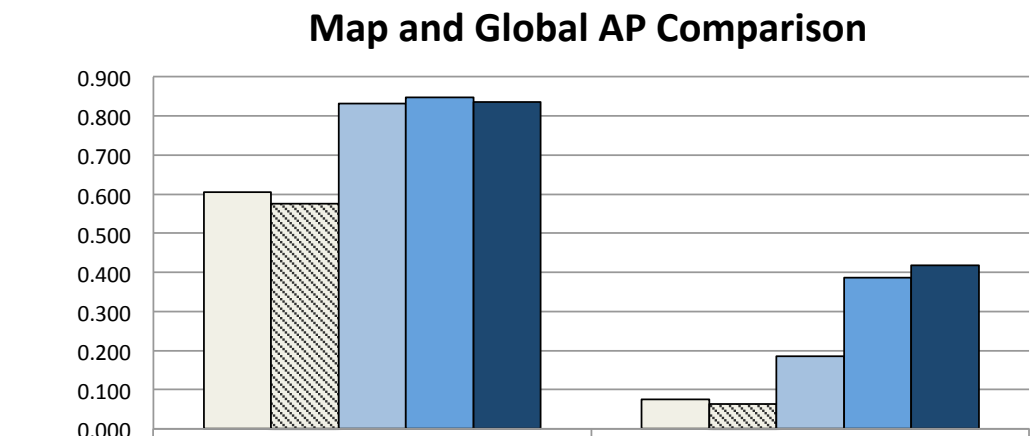
Section 2.2.3. In order to make a thorough comparison, we report the tracing results of each method using both MAP and Global AP measures in this chapter.

6.2.3 Result Analysis

The left part of Figure 6.1 illustrates the Mean Average Precision (MAP) values using VSM, LSI, and the Tracing Network (TN) with different negative link sampling ratio, i.e. $k = 1$ (TN_Neg1), $k = 5$ (TN_Neg5), $k = 10$ (TN_Neg10). A Friedman test identified a statistically significant difference among the APs associated with these methods ($\mathcal{X}^2(4) = 88.225$, $p < 2.2e^{-16}$). We observe that the tracing network performs significantly better than both VSM and LSI regardless of the value of k. When we change the value of k, however, the MAP does not demonstrate significant differences. The p value of the pairwise Wilcoxon signed ranks tests with Bonferroni adjustments is shown in Table 6.1. Considering the way the MAP metric is calculated, this result implies that the negative link sampling does not strongly affect the *ranking* of the target artifacts for each individual source artifact.

When we delve deeper into the actual similarity scores assigned by TN_Neg1, TN_Neg5, TN_Neg10 for each artifact pair, we observe prominent differences. To illustrate these differences, Table 6.2 provides the top target artifacts returned for the source *SSRS.11061*. Each column presents the scores assigned to the target artifacts by the TN with their corresponding k values. The linked target artifact *SSDD.3530* is ranked on the top of the list in all three settings. Nevertheless, when we increase the negative link sampling ratio, TN_Neg5 and TN_Neg10 are able to assign much smaller scores to the other target artifacts that are **not** linked to this source.

We hypothesize that TN_Neg5 and TN_Neg10 are better at identifying negative links as they have seen more negative examples during training. This hypothesis is confirmed by their Global AP measures. As the right hand side of Figure 6.1 shows, TN_Neg5 and



	MAP	Global AP
VSM	0.605	0.074
LSI	0.576	0.064
TN_Neg1	0.832	0.185
TN_Neg5	0.847	0.386
TN_Neg10	0.835	0.419

Figure 6.1. The Comparison of MAP and Global AP on PTC Dataset

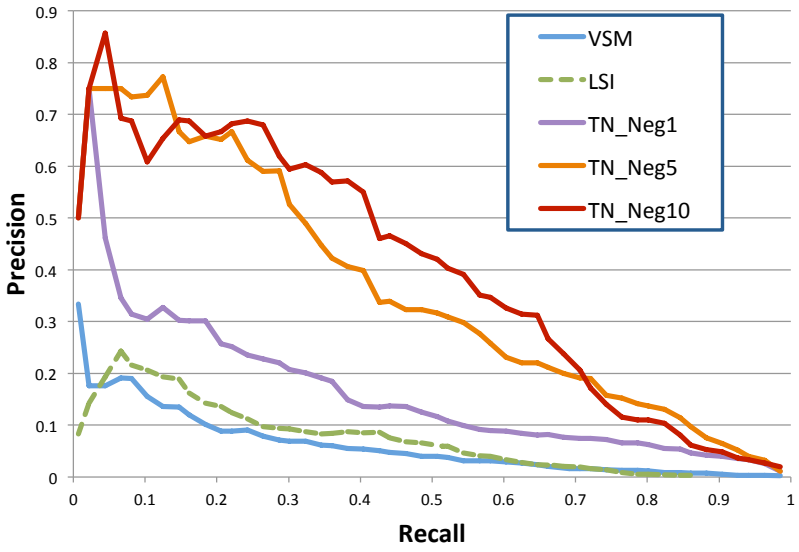


Figure 6.2. Precision and Recall Curve on testing dataset

TABLE 6.1

MAP PAIRWISE COMPARISONS USING WILCOXON RANK SUM TEST

	LSI	TN_Neg1	TN_Neg5	TN_Neg10
TN_Neg1	1.90E-07	-	-	-
TN_Neg5	2.30E-08	1	-	-
TN_Neg10	9.40E-08	1	1	-
VSM	1	2.10E-06	3.20E-07	1.70E-06

TABLE 6.2

TOP TARGET ARTIFACTS RETURNED BY TN USING DIFFERENT k

Target Artifact ID	TN_Neg1	TN_Neg5	TN_Neg10	Answer
SSDD.3530	0.998	0.961	0.993	Link
SSDD.3552	0.996	0.683	0.436	Non-link
SSDD.3523	0.991	0.857	0.587	Non-link
SSDD.3561	0.981	0.200	0.097	Non-link
SSDD.716	0.963	0.306	0.015	Non-link

Note: the target artifacts are ranked according to the scores returned by TN_Neg1.

TN_Neg10 both improve the Global AP over VSM, LSI and TN_Neg1 substantially. The Precision-Recall Curve also clearly demonstrates the advantage with increased value of k. As shown in Figure 6.2, in both cases ($k = 5, 10$), the tracing network is less sensitive to the setting of the threshold of the minimal score returned to the user, and it increases the precision at all levels of recall. Such an advantage is especially important for the usage scenario we introduced earlier in which the users expect all the valid trace links from the dataset to be returned at the top of a single list.

Nevertheless, as we increase the number of negative links included in the training process, the training time is prolonged k times. More importantly, as we further increase k, the tracing performance begins to deteriorate due to the dominant number of negative links. To balance the efficiency and the performance, we therefore set $k = 10$ for all other experiments reported in this chapter.

6.3 Applying Tracing Network to Other Datasets

A practical tracing solution needs to work well when applied to a variety of projects and should ideally address the tracing needs for both large and small datasets. In Chapter 5, we have demonstrated that the tracing network can achieve significantly better tracing results than VSM and LSI for large datasets. In Section 6.2, we have further improved its performance with the negative link sampling technique. In this section, we aim to explore whether the tracing network can bring enhancements on smaller traceability datasets that are more common for projects with refined scope or for projects in their earlier phases.

6.3.1 Data Preparation

We gather two additional datasets from domains of the Positive Train Control (PTC) and the Electronic Health Records (EHR) system. The PTC dataset represents a different, smaller subsystem of the PTC project discussed in Chapter 5. To differentiate between those two datasets, from now on we refer to this new dataset as “PTC Small” and the one

TABLE 6.3

CHARACTERISTICS OF THREE TRACING DATASETS

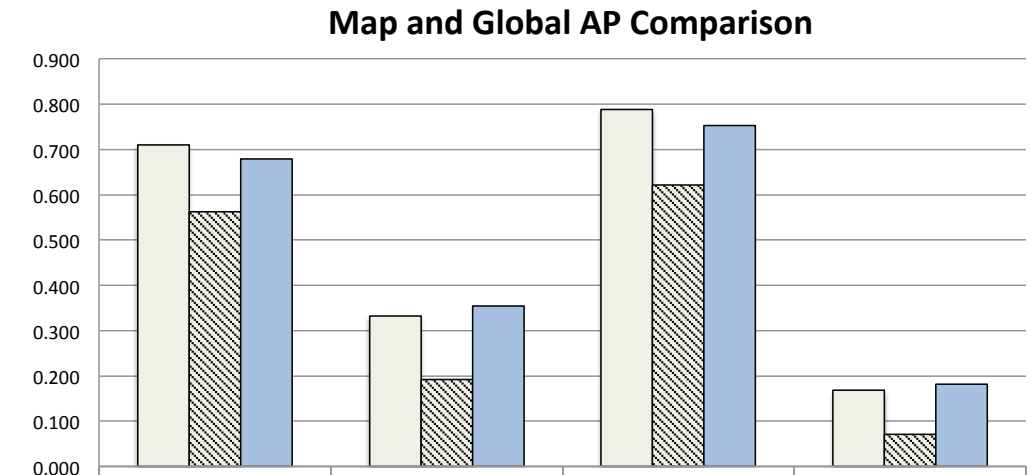
Dataset Name	No. of Src. Artifact	No. of Trg. Artifact	Src. Avg. Token	Trg. Avg. Token	No. of Links
PTC Large	1,651	466	33	99	1,387
PTC Small	101	241	30	51	241
EHR	1,064	116	29	25	586

used in Chapter 5 and Section 6.2 as "PTC Large". The PTC Small dataset contains 101 Software Requirements (62 system requirements and 39 subsystem requirements) as source queries, 241 System Design Description as target documents, and has 404 associated trace links. The EHR dataset includes 1,064 regulations developed by the Certification Commission for Health Information Technology (CCHIT) [24], 116 requirements extracted from USA Veterans Health Care system [144], and 586 associated trace links. The characteristics of those three datasets are summarized in Table 6.3.

During the word embeddings pre-training phase, we use the same PTC domain corpus described in Section 5.4.1 for the PTC Small datasets. For the EHR dataset, we used 74.6MB of clean text that we extracted from 1.05GB pdf files describing the domain of health IT systems. Those PDF files were obtained through scraping documents from HealthIT.gov.

6.3.2 Tracing Network Performance

In the experiment setting, we use the same split ratio to divide the datasets, i.e. 80% of the datasets are randomly selected for training, 10% for choosing the hyper-parameters, and 10% for evaluation. For the Tracing Network (TN) configuration, we fix the RNN Unit Type as BI-GRU and negative link sampling ratio k as 10. We then use the same



	PTC Small		EHR	
	MAP	Global MAP	MAP	Global MAP
VSM	0.710	0.332	0.789	0.169
LSI	0.562	0.192	0.622	0.071
TN	0.679	0.355	0.753	0.181

Figure 6.3. Comparison of average MAP and Global AP using VSM, LSI and TN on dataset PTC Small and EHR

procedure described in Section 5.4.2 for tuning other hyper-parameters. We evaluate the performance of the TN on the testing sets and calculate the MAP and Global AP measures. For the baseline methods VSM and LSI, we also use the best configurations customized for those two datasets in order to make a fair comparison. To reduce the effect of random data selection, we split the datasets and repeat this process ten times. We report the average results obtained from these ten repetitions in Figure 6.3.

We observe the same trends for both PTC Small and EHR datasets. LSI performs considerably worse than all other methods – achieving the lowest MAP and Global AP scores. The deteriorated performance of LSI may be ascribed to the fact that there are not enough co-occurrences of terms to capture the latent semantic associations between them when the dataset is small [89].

Although TN obtains slightly better results at Global AP measure than VSM, and VSM

TABLE 6.4

TOP 10 CANDIDATE LINKS RETURNED BY DIFFERENT TRACING METHODS

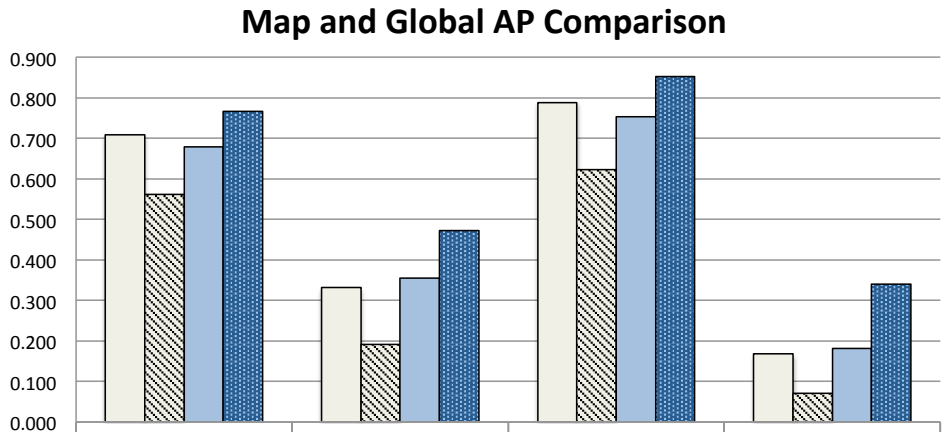
(a) Top VSM links

Rank	VSM Score	Answer	TN Score
1	0.557	Link	0.789
2	0.550	Non-Link	0.014
3	0.508	Link	0.691
4	0.500	Link	0.919
5	0.487	Non-Link	0.674
6	0.480	Link	0.650
7	0.470	Non-Link	0.076
8	0.468	Non-Link	0.003
9	0.466	Non-Link	0.185
10	0.443	Non-Link	0.083

(b) Top TN links

Rank	TN Score	Answer	VSM Score
1	0.954	Link	0.323
2	0.950	Link	0.065
3	0.949	Link	0.115
4	0.946	Non-Link	0.236
5	0.943	Link	0.314
6	0.942	Non-Link	0.034
7	0.935	Link	0.187
8	0.932	Non-Link	0.000
9	0.930	Non-Link	0.000
10	0.930	Link	0.140

performs better than TN at MAP measure, these variances are not significant. However, when we closely inspect the ranked list of generated trace links, we observe obvious discrepancies in the results obtained by those two methods. We present the top 10 trace link candidates returned by VSM and TN for the EHR dataset in Table 6.4a and 6.4b respectively. Within the top ten candidates, 60% of VSM links and 40% of TN links are incorrect. However, we observe that the VSM and TN methods demonstrate a somewhat complementary phenomenon; i.e., in both cases, the other method is able to eliminate most of the incorrectly retrieved candidate links by assigning minimal scores to them (highlighted in red). In Chapter 5, we have demonstrated that, for a large dataset, TN is able to capture the semantic information of the artifacts, and utilize such information to construct effective as-

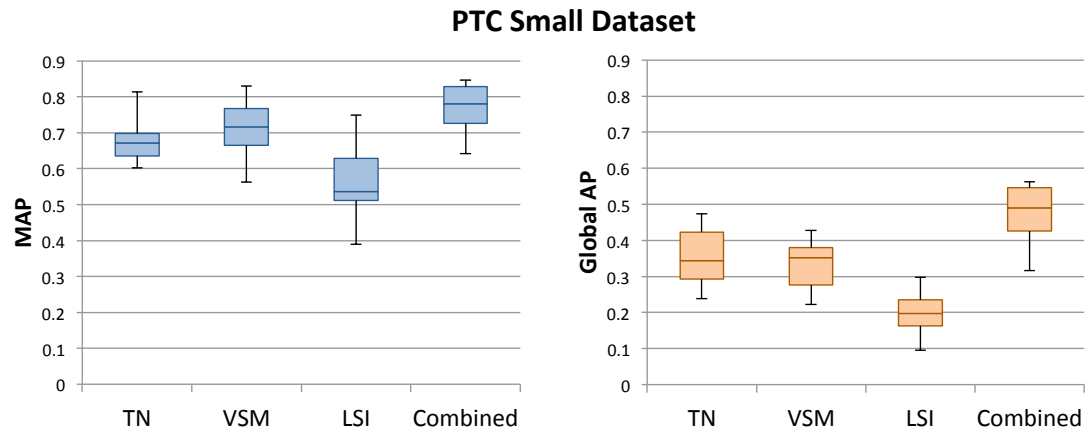


	PTC Small		EHR	
	MAP	Global MAP	MAP	Global MAP
VSM	0.710	0.332	0.789	0.169
LSI	0.562	0.192	0.622	0.071
TN	0.679	0.355	0.753	0.181
Combined	0.766	0.472	0.853	0.340

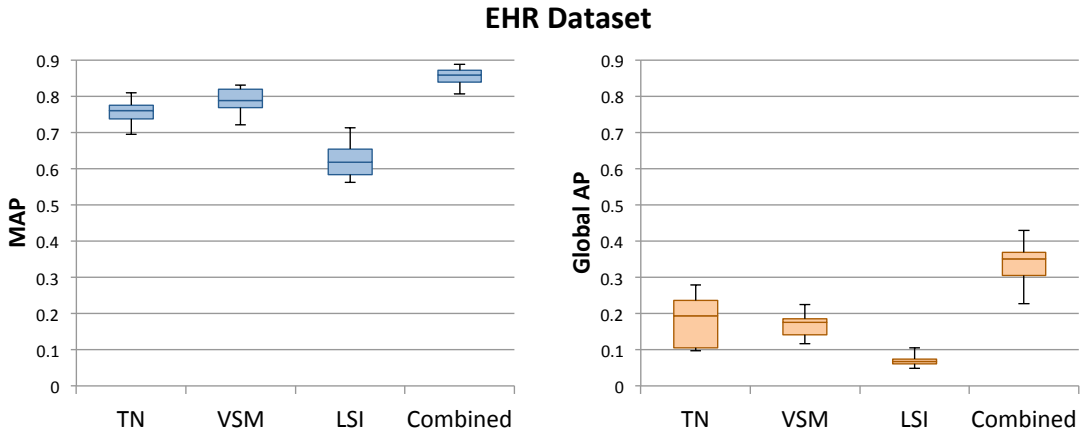
Figure 6.4. Comparison of average MAP and Global AP using all four methods on dataset PTC Small and EHR

sociations between artifacts. Nevertheless, when the number of trace links is not sufficient, we speculate that TN only captures a limited amount of semantic information, and it fails to eliminate some associations that do not contribute to trace links. In such cases, VSM serves as a useful supplemental method to eliminate some of the incorrect associations. On the other hand, the VSM method mistakenly links many artifacts with common terms. Those links can be recognized, and eliminated by TN if it has seen similar examples during the training process.

Based on these observations, we propose a **combined** method that effectively overcomes some of the drawbacks of both VSM and TN on small datasets. This combined method assigns each link candidate a similarity score using the product of VSM and TN scores and then ranks the link candidates accordingly. The comparison of the combined method with all other methods is also based on ten iterations. We report the average result achieved by each method at MAP and Global AP in Figure 6.4. We also report the



(a)



(b)

Figure 6.5. The distribution of MAP (left) and Global AP (right) of four Tracing Methods during each repetition on dataset PTC Small (a) and EHR (b).

distribution of these two measures during each repetition in Figure 6.5 as box plots.

The combined method performs significantly better than both VSM and TN in terms of the MAP measure on EHR datasets, and significantly better than both VSM and TN with the Global AP measure on both datasets using the pairwise Wilcoxon signed ranks tests with Bonferroni adjustments ($p < 0.05$). Especially in the case of the Global AP measure, the combined method achieved 87.8% improvement on the EHR dataset. These results clearly demonstrate the benefit of combining VSM and TN on small tracing datasets. However, we have not observed similar advantages when using the combined method on

the PTC Large dataset. This is expected because when the dataset is sufficiently large, the TN method itself can adequately distinguish positive trace links from negative links through training.

Based on the datasets we studied, we conclude that the VSM method benefits when combined with TN for small datasets if the training data is available. For large datasets, however, using TN on its own appears to yield strong performance. In both cases, TN has proved especially helpful for finding missing links that were initially overlooked by the tracing experts. Consequently, in practice the project stakeholders should consider their resources, the nature of their projects, and the required tracing accuracy to make decisions about which tracing method to use. VSM is an effective and convenient method to construct trace links for small datasets. It is easy to implement and does not require any training data. Nevertheless, when more trace links become available as the project proceeds, using TN to augment or replace VSM could offer substantial benefits through increasing accuracy of the generated trace links.

6.4 Moving Forward: Towards More Generalizable Tracing Solutions

There are still many open challenges to be addressed in order to fully leverage semantic analysis using the deep learning based approach. Despite the fact that the current TN is able to achieve significantly better MAP and Global AP scores on large traceability datasets, its advantage over the VSM method on small datasets is less obvious. Nevertheless, there are several ways ahead that can potentially improve the performance of the deep learning based methods on small datasets and consequently lead to more generalizable tracing solutions. In Section 6.3, we have demonstrated that combining TN with VSM is effective for enhancing tracing results on a small dataset. In this section, we propose other techniques that might improve the current tracing methods. We also discuss the presented challenges with adopting those techniques.

6.4.1 Sentence Embedding Pre-Training

Sentence embedding aims to learn distributed sentence representations that can represent the semantic information of whole sentences in supervised or unsupervised manners. In this regard, the TN method proposed in Chapter 5 can also be considered as one supervised model to train sentence embeddings – it uses BI-GRU to encode the semantic of the sentences in one artifact into a single vector representation, and the objective of the training is to use such artifact representations for trace link evaluation. However, as introduced in Section 6.1, the number of available links are limited for many traceability datasets. Similar with how the tracing network can benefit from pre-training word embeddings from unlabeled data, we therefore are interested if the tracing network can also achieve improved performance with pre-trained sentence embedding models without requiring large amount of trace links.

Several different models has been proposed to train sentence embeddings using unlabeled data [77, 35]. But it is unknown which architectures or objectives would yield the best sentence representations. Two of the commonly used ones include:

- **Skip-thought** aims to learn distributed representations for sentences with a encoder-decoder model [77]. The idea is similar to the Skip-gram model proposed in Word2Vec [101], but applied on the sentence level. The encoder maps words from one sentence into a sentence vector, and then the decoder conditions on this vector to predict words in surrounding sentences. R. Kiros et al. have explored several options for the encoder and decoder models in their work [77], including RNN, LSTM, and GRU. They finally chose GRU as its encoder and decoder models since GRU yields top performance and is conceptually simpler.
- **Auto-encoder** is also based on the encoder-decoder framework. Different from Skip-thought that uses one sentence to predict its surrounding sentences, the objective of auto-encoder is to utilize the encoded sentence vectors to predict words in the sentence itself. A. M. Dai et al. selected the LSTM model as their encoder and decoder for pre-training purposes [35]. They found that the pre-trained LSTM becomes more stable to train and generalizes better for subsequent tasks such as sentiment analysis and document classification.

6.4.2 Challenges

As with our initial exploration, we have examined the performance of both Skip-thought and Auto-encoder with BI-GRU model as the encoder and decoder, and utilized the pre-trained BI-GRU in our tracing network for datasets PTC Big, PTC Small and EHR. The corpora for the pre-training are the same domain document collections that we used for training the word representations. However, we did not observed satisfactory results in our informal experiments. The performance of the tracing network was not improved using the pre-trained BI-GRU compared to using BI-GRU with randomly initialized parameters. We suggest two plausible explanations:

(1). It is possible that the objective of neither Skip-thought nor Auto-encoder captures the essence of the sentence semantics that are useful for the tracing task. Hill et al. suggest that the actual task type is decisive on how to choose the appropriate pre-training approach in their recent work [68]. Therefore, in order to benefit from pre-training the sentence embedding models, it is important to choose the structure that the objective of the pre-training step would actually coincide with the trace link generation task.

(2). The hidden dimension of BI-GRU model in Skip-thought and Auto-encoder needs to be set large enough to achieve an acceptable performance on their objective of reconstructing sentences. But such a complex model cannot be properly retrained later for the tracing task because there are only limited number of trace links available.

We believe the above two challenges are interrelated. As the objective of the pre-training gets close enough to the tracing task, the required number of trace links would be minimized to retrain the models for the tracing network. In the future, we plan to explore other supervised and unsupervised sentence embedding models or variations that would contribute to extracting the semantic of software artifact for tracing purpose.

6.4.3 Other Directions

Current deep learning techniques are not a silver bullet. It may be difficult to design and train a deep learning model that is sophisticated enough to capture all the nuances of complex artifacts in the software engineering domain. We therefore believe it is also important to explore hybrid solutions that leverage deep learning techniques in conjunction with other approaches that deploy prior knowledge about sentence syntax and/or domain concepts. For example, instead of using a general sequence model such as RNN to encode the software artifacts, it might be possible to utilize or design a structured model that is more suited to capture the semantics of software artifacts, such as models that are better at processing long and complex sentences [15]. On the other hand, software engineering projects produce a wide variety of artifacts. These artifacts potentially contain rich knowledge about the projects and their target domains. For instance, project glossaries describe the key terms used in the project in natural language, while domain models connect domain concepts through meaningful relationships with pre-defined meta-models. The implicit relations between terms or sentences provided by those data can be utilized to train more accurate word/phrase or sentence representations in the context of specific domains [90]. Meanwhile based on our initial analysis we believe it would be worthwhile investigating hybrid solutions that leverage heuristic approaches in conjunction with deep learning.

6.5 Conclusion

In the chapter, we have carried out several initial investigations to improve the current tracing network method. The negative link sampling technique has demonstrated to bring notable augmentation to the tracing network at the Global AP measure on large tracing dataset. On the small dataset, the tracing network provides substantial assistance to the state-of-the-art VSM method and consequently leads to better MAP and Global AP measures. At the same time, we have also perceived great potentials and challenges to

further enhance the tracing network with sentence embedding techniques. We suggest several directions to follow in the future to meet the goal of more semantically enhanced and generalizable tracing solutions.

CHAPTER 7

CONCLUSION

This dissertation presents my major contributions towards semantically enhanced tracing solutions. I have proposed a series of methods that integrate different levels of semantic information from the project domain and/or software artifacts. I have implemented those methods and carried out extensive evaluations on datasets of trace links between regulations, requirements, and design documents. The innovations and limitations of each method are discussed in detail. As my research goal in a long run, I aim to seek more effective approaches to extract and utilize software artifact semantics across projects and domains, and ultimately provide sufficient support for software stakeholders' tracing needs.

7.1 Contributions

Firstly, Chapter 1 introduces the important concepts in the context of software traceability. It reveals the major barriers that hinder the state-of-the-art tracing algorithms from achieving satisfactory results and from being adopted in industry. It then summarizes the goals of this dissertation and offers an overview of the content in each chapter.

Chapter 2 shows that domain knowledge captured in the form of ontology can be utilized to augment the tracing queries. It demonstrates that such domain ontology is effective for connecting artifacts that contain related domain concepts. It also indicates that the domain ontology created from trace links in one project can be successfully applied to other projects from the same domain. This chapter also provides a close comparison with two other trace query augmentation based methods and a detailed summary of their costs,

benefits and constraints. This work was published in the *Journal of Empirical Software Engineering* [62].

The work described in Chapter 3 and 4 presents a novel heuristic based approach that approximate how human experts generate trace links. It effectively integrates the knowledge about both the target domain and software artifacts into the automated tracing process. The proposed solution, DoCIT, processes the software artifacts, reasons over their associations in both term-level and sentence-level, creates trace links with handcrafted link heuristic rules, and finally generates explanations for its decisions in natural language format. DoCIT is one of the few recent works that utilize sophisticated natural language processing and knowledge representation techniques to understand and partially replicate the way in which a human performs the tracing task [27]. This work was published in the *Proceedings of the International Conference on Automated Software Engineering* [58] and in the *Proceedings of the International Conference on Requirements Engineering* [59].

Finally, Chapter 5 and 6 propose a tracing solution that adopts deep learning techniques to overcome the major limitations of DoCIT. The experiments in these two chapters demonstrate significant advantages of using the proposed tracing network architecture for large tracing datasets. Even for small datasets, combining the tracing network with VSM has proven beneficial to enhance the tracing performance. These results clearly show the potential of leveraging deep learning based methods for the tracing task. This work also poses fresh challenges for future research on how to achieve more general tracing solutions in order to tackle the tracing tasks from projects with different characteristics. Part of this work was published in the *Proceedings of the International Conference on Software Engineering* [61].

7.2 Future Work

7.2.1 Ontology and Heuristic Learning

The tracing solutions presented from Chapter 2 to 4 all requires different degrees of manual input, ranging across the domain ontology, trace link heuristics, as well as rationale generation heuristics. Creating these inputs can be very costly especially for large projects in complex domains. To ease the burden of manually generating these data, it is worth investigating automated or semi-automated approaches that learn the relevant knowledge from existing documents. In our early work, we have demonstrated that the domain ontology can be learned from domain corpora and existing trace links with NLP methods [56]. In the future, we can also adopt search based optimization methods and bootstrapping techniques to improve the learning process of the ontology and to suggest new trace link heuristics.

7.2.2 Sentence Representation Pre-Training and Domain Adaptation

Chapter 5 proposes the tracing network structure that utilizes BI-GRU model to extract the sentence semantics. The BI-GRU is initialized randomly and it learns how to represent sentences through artifact pairs associated by trace links. However, the BI-GRU may not be sufficiently trained due to the limited number of trace links. Therefore, in the future, we plan to explore whether the tracing network can utilize pre-trained sentence representations to improve its performance, especially on smaller datasets. In Chapter 6, we discuss our initial observation on pre-training the BI-GRU model with Skip-thoughts and Auto-encoder. We will explore other sentence representation pre-training techniques that can adequately capture the sentence semantics for tracing rationales. On the other hand, we are also interested in examining if a trained tracing network can be successfully adapted for new projects with minimal effort. We plan to evaluate the effectiveness and effort for re-training the tracing network and its components for projects from the same and/or different

domains.

7.2.3 Combining Heuristic-Based and Deep Learning Approaches

Despite the rapid progress of current deep learning research, it may be difficult to design and train a deep learning model that is capable of understanding and reasoning over complex artifacts from software engineering projects. Consequently, it is important to look for hybrid solutions that leverage heuristic approaches in conjunction with deep learning. Both methods offer practical advantages and unique challenges. Combining them might bring opportunities to maximize the tracing result with currently available resources. We therefore need to investigate the deep learning models that can effectively store and retrieve the structured input for utilizing domain knowledge and heuristics during the tracing process.

7.2.4 Putting Trace Engine in the Loop

Finally, we believe that the automated tracing methods cannot replace the human's role in the process of trace link generation and evaluation. In the end, the tracing solution needs to be integrated into stakeholders' activities in real industrial settings. Those methods therefore need to understand the context of the tracing task and to accept feedback from the users. Chapter 4 illustrates our preliminary work on communicating the tracing rationales to the users to support their trace link evaluation tasks. In the future, we plan to investigate other aspects of the tracing task, to increase the level of assistance provided by the tracing methods, and to seamlessly integrate the tracing methods into the software stakeholders' working process.

BIBLIOGRAPHY

1. S. L. Abebe and P. Tonella. Extraction of domain concepts from the source code. *Sci. Comput. Program.*, 98:680–706, 2015. doi: 10.1016/j.scico.2014.09.012. URL http://dx.doi.org/10.1016/j.scico.2014.09.012.

2. R. Agrwal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Intn'l Conf. on Very Large Data Bases (VLDB'94)*, pages 487–499, Santiago, Chile, September 1994.

3. G. Antoniol, G. Canfora, A. de Lucia, and G. Casazza. Information retrieval models for recovering traceability links between code and documentation. In *ICSM '00: Proceedings of the International Conference on Software Maintenance (ICSM'00)*, page 40, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0753-0.

4. G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Trans. Softw. Eng.*, 28(10): 970–983, Oct. 2002. ISSN 0098-5589. doi: 10.1109/TSE.2002.1041053. URL http://dx.doi.org/10.1109/TSE.2002.1041053.

5. N. Assawamekin, T. Sunetnanta, and C. Pluempitiwiriyawej. Ontology-based multiperspective requirements traceability framework. *Knowl. Inf. Syst.*, 25(3):493–522, 2010. URL http://dblp.uni-trier.de/db/journals/kais/kais25.html#AssawamekinSP10.

6. H. U. Asuncion, A. Asuncion, and R. N. Taylor. Software traceability with topic modeling. In J. Kramer, J. Bishop, P. T. Devanbu, and S. Uchitel, editors, *International Conference on Software Engineering*, pages 95–104. ACM, 2010. ISBN 978-1-60558-719-6.

7. M. A. Babar and I. Gorton. A tool for managing software architecture knowledge. In *Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*, SHARK-ADI '07, pages 11–, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2951-8. doi: http://dx.doi.org/10.1109/SHARK-ADI.2007.1. URL http://dx.doi.org/10.1109/SHARK-ADI.2007.1.

8. R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999. ISBN 0-201-39829-X.

9. D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL http://arxiv.org/abs/1409.0473.

10. I. G. Y. Bengio and A. Courville. Deep learning. Book in preparation for MIT Press, 2016. URL http://www.deeplearningbook.org.

11. Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, pages 437–478. Springer, 2012.

12. Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

13. K. H. Bennett and V. T. Rajlich. Software maintenance and evolution: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 73–87, New York, NY, USA, 2000. ACM. ISBN 1-58113-253-0. doi: http://doi.acm.org/10.1145/336512.336534.

14. B. Berenbach, D. Gruseman, and J. Cleland-Huang. Application of just in time tracing to regulatory codes. In *Proceedings of the Conference on Systems Engineering Research*, Holbroken, NJ, 2010. Stevens Institute of Technology. doi: http://doi.ieeecomputersociety.org/10.1109/RE.2009.20.

15. S. R. Bowman, C. D. Manning, and C. Potts. Tree-structured composition in neural networks without tree-structured architectures. In *Proceedings of the 2015th International Conference on Cognitive Computation: Integrating Neural and Symbolic Approaches - Volume 1583*, COCO'15, pages 37–42, Aachen, Germany, Germany, 2015. CEUR-WS.org. URL http://dl.acm.org/citation.cfm?id=2996831.2996836.

16. T. D. Breaux and A. Rao. Formal analysis of privacy requirements specifications for multi-tier applications. In *21st IEEE International Requirements Engineering Conference, RE 2013, Rio de Janeiro-RJ, Brazil, July 15-19, 2013*, pages 14–20, 2013. doi: 10.1109/RE.2013.6636701. URL http://dx.doi.org/10.1109/RE.2013.6636701.

17. T. D. Breaux, A. I. Antón, K. Boucher, and M. Dorfman. Legal requirements, compliance and practice: An industry case study in accessibility. In *RE*, pages 43–52, 2008.

18. T. D. Breaux, A. I. Antón, and J. Doyle. Semantic parameterization: A process for modeling domain descriptions. *ACM Trans. Softw. Eng. Methodol.*, 18(2):5:1–5:27, Nov. 2008. ISSN 1049-331X. doi: 10.1145/1416563.1416565. URL http://doi.acm.org/10.1145/1416563.1416565.

19. A. Z. Broder, M. Fontoura, E. Gabrilovich, A. Joshi, V. Josifovski, and T. Zhang. Robust classification of rare queries using web knowledge. In *SIGIR '07: Proceedings*

of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, pages 231–238, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-597-7. doi: http://doi.acm.org/10.1145/1277741.1277783.

20. M. Buckland and F. Gey. The relationship between recall and precision. *Journal of the American society for information science*, 45(1):12, 1994.

21. J. E. Burge and D. C. Brown. Software engineering using rationale. *Journal of Systems and Software*, 81(3):395–413, 2008.

22. R. Capilla, F. Nava, S. Pérez, and J. C. Dueñas. A web-based tool for managing architectural design decisions. *SIGSOFT Softw. Eng. Notes*, 31, Sept. 2006. ISSN 0163-5948. doi: http://doi.acm.org/10.1145/1163514.1178644. URL `http://doi.acm.org/10.1145/1163514.1178644`.

23. C. Carpineto and G. Romano. A survey of automatic query expansion in information retrieval. *ACM Computing Surveys (CSUR)*, 44(1):1, 2012.

24. CCHIT. Certification commission for health information technology, 2015. URL `http://www.cchit.org/`.

25. K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

26. J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

27. J. Cleland-Huang and J. Guo. Towards more intelligent trace retrieval algorithms. In B. Turhan, A. B. Bener, Ç. Meriçli, A. V. Miranskyy, and L. L. Minku, editors, *3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, RAISE 2014, Hyderabad, India, June 3, 2014*, pages 1–6. ACM, 2014. ISBN 978-1-4503-2846-3. doi: 10.1145/2593801.2593802. URL `http://doi.acm.org/10.1145/2593801.2593802`.

28. J. Cleland-Huang, C. K. Chang, and M. Christensen. Event-based traceability for managing evolutionary change. *IEEE Trans. Softw. Eng.*, 29(9):796–810, 2003. ISSN 0098-5589. doi: http://dx.doi.org/10.1109/TSE.2003.1232285.

29. J. Cleland-Huang, R. Settimi, C. Duan, and X. Zou. Utilizing supporting evidence to improve dynamic requirements traceability. In *13th IEEE International Conference on Requirements Engineering (RE 2005), 29 August - 2 September 2005, Paris, France*, pages 135–144, 2005. doi: 10.1109/RE.2005.78. URL `http://dx.doi.org/10.1109/RE.2005.78`.

30. J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc. The detection and classification of non-functional requirements with application to early aspects. In *RE*, pages 36–45, 2006.

31. J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc. Automated detection and classification of non-functional requirements. *Requir. Eng.*, 12(2):103–120, 2007. ISSN 0947-3602. doi: http://dx.doi.org/10.1007/s00766-007-0045-1.

32. J. Cleland-Huang, M. Czauderna, Adam fand Gibiec, and J. Emenecker. A machine learning approach for tracing regulatory codes to product specific requirements. In *ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pages 155–164, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-719-6. doi: http://doi.acm.org/10.1145/1806799.1806825.

33. J. Cleland-Huang, O. Gotel, J. H. Hayes, P. Mäder, and A. Zisman. Software traceability: trends and future directions. In J. D. Herbsleb and M. B. Dwyer, editors, *Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014*, pages 55–69. ACM, 2014. ISBN 978-1-4503-2865-4. doi: 10.1145/2593882.2593891. URL http://doi.acm.org/10.1145/2593882.2593891.

34. D. Cuddeback, A. Dekhtyar, and J. H. Hayes. Automated requirements traceability: the study of human analysts. In *RE'10: Proceedings of the IEEE International Requirements Engineering Conference*. IEEE, 2010.

35. A. M. Dai and Q. V. Le. Semi-supervised sequence learning. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, NIPS'15, pages 3079–3087, Cambridge, MA, USA, 2015. MIT Press. URL http://dl.acm.org/citation.cfm?id=2969442.2969583.

36. T. Dasgupta, M. Grechanik, E. Moritz, B. Dit, and D. Poshyvanyk. Enhancing software traceability by automatically expanding corpora with relevant documentation. In *2013 IEEE International Conference on Software Maintenance, Eindhoven, The Netherlands, September 22-28, 2013*, pages 320–329, 2013. doi: 10.1109/ICSM.2013.43. URL http://dx.doi.org/10.1109/ICSM.2013.43.

37. A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. Enhancing an artefact management system with traceability recovery features. In *Proceedings of the 20th IEEE International Conference on Software Maintenance*, ICSM '04, pages 306–315, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2213-0. URL http://dl.acm.org/citation.cfm?id=1018431.1021438.

38. T. Dietrich, J. Cleland-Huang, and Y. Shin. Learning effective query transformations for enhanced requirements trace retrieval. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*, pages 586–591, 2013. doi: 10.1109/ASE.2013.6693117. URL http://dx.doi.org/10.1109/ASE.2013.6693117.

39. A. Egyed. A scenario-driven approach to trace dependency analysis. *IEEE Trans. Softw. Eng.*, 29(2):116–132, 2003. ISSN 0098-5589. doi: http://dx.doi.org/10.1109/TSE.2003.1178051.

40. D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

41. D. Falessi, G. Cantone, and G. Canfora. Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques. *IEEE Trans. Software Eng.*, 39(1):18–44, 2013. doi: 10.1109/TSE.2011.122. URL http://doi.ieeecomputersociety.org/10.1109/TSE.2011.122.

42. *Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices*. Food and Drug Administration, 2005.

43. E. Gabrilovich, A. Broder, M. Fontoura, A. Joshi, V. Josifovski, L. Riedel, and T. Zhang. Classifying search queries using the web as a source of knowledge. *ACM Trans. Web*, 3(2):1–28, 2009. ISSN 1559-1131. doi: http://doi.acm.org/10.1145/1513876.1513877.

44. R. Gacitua and P. Sawyer. Ensemble methods for ontology learning - an empirical experiment to evaluate combinations of concept acquisition techniques. In *ACIS-ICIS*, pages 328–333, 2008.

45. R. Gacitua, P. Sawyer, and P. Rayson. A flexible framework to experiment with ontology learning techniques. *Knowl.-Based Syst.*, 21(3):192–199, 2008.

46. Gervasi and D. Zowghi. Supporting traceability through affinity mining. In *Conference on Requirements Engineering*, pages 143–152, 2014.

47. M. Gibiec, A. Czauderna, and J. Cleland-Huang. Towards mining replacement queries for hard-to-retrieve traces. In *ASE '10: Proceedings of the IEEE/ACM international conference on Automated software engineering*, pages 245–254, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0116-9. doi: http://doi.acm.org/10.1145/1858996.1859046.

48. O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, and J. Maletic. The grand challenge of traceability (v1.0). In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 343–409. Springer London, London, 2012. ISBN 978-1-4471-2239-5. doi: 10.1007/978-1-4471-2239-5_16. URL http://dx.doi.org/10.1007/978-1-4471-2239-5_16.

49. O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, J. I. Maletic, and P. Mäder. Traceability fundamentals. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability.*, pages 3–22. Springer, 2012. ISBN 978-1-4471-2238-8. doi: 10.1007/978-1-4471-2239-5_1. URL http://dx.doi.org/10.1007/978-1-4471-2239-5_1.

50. O. C. Gotel and C. Finkelstein. An analysis of the requirements traceability problem. In *Requirements Engineering, 1994., Proceedings of the First International Conference on*, pages 94–101. IEEE, 1994.

51. O. C. Z. Gotel and A. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of the First IEEE International Conference on Requirements Engineering, ICRE '94, Colorado Springs, Colorado, USA, April 18-21, 1994*, pages 94–101, 1994. doi: 10.1109/ICRE.1994.292398. URL http://dx.doi.org/10.1109/ICRE.1994.292398.

52. A. Grando and R. Schwab. Building and evaluating an ontology-based tool for reasoning about consent permission. In *AMIA Annual Symposium Proceedings*, pages 514–523, 2013.

53. M. Grando, A. Boxwala, R. Schwab, and N. Alipanah. Ontological approach for the management of informed consent permissions. In *Healthcare Informatics, Imaging and Systems Biology (HISB), 2012 IEEE Second International Conference on*, pages 51–60, Sept 2012. doi: 10.1109/HISB.2012.19.

54. T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

55. G. Guizzardi. Theoretical foundations and engineering tools for building ontologies as reference conceptual models. *Semantic Web*, 1(1-2):3–10, 2010. doi: 10.3233/SW-2010-0015. URL http://dx.doi.org/10.3233/SW-2010-0015.

56. J. Guo. Ontology learning and its application in software-intensive projects. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume*, pages 843–846, 2016. doi: 10.1145/2889160.2889264. URL http://doi.acm.org/10.1145/2889160.2889264.

57. J. Guo, J. Cleland-Huang, and B. Berenbach. Foundations for an expert system in domain-specific traceability. In *21st IEEE International Requirements Engineering Conference, RE 2013, Rio de Janeiro-RJ, Brazil, July 15-19, 2013*, pages 42–51. IEEE Computer Society, 2013. ISBN 978-1-4673-5765-4. doi: 10.1109/RE.2013.6636704. URL http://dx.doi.org/10.1109/RE.2013.6636704.

58. J. Guo, N. Monaikul, C. Plepel, and J. Cleland-Huang. Towards an intelligent domain-specific traceability solution. In *ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014*, pages 755–766, 2014. doi: 10.1145/2642937.2642970. URL http://doi.acm.org/10.1145/2642937.2642970.

59. J. Guo, N. Monaikul, and J. Cleland-Huang. Trace links explained: An automated approach for generating rationales. In *23rd IEEE International Requirements Engineering Conference, RE 2015, Ottawa, ON, Canada, August 24-28, 2015*, pages

202–207, 2015. doi: 10.1109/RE.2015.7320423. URL http://dx.doi.org/10.1109/RE.2015.7320423.

60. J. Guo, M. Rahimi, J. Cleland-Huang, A. Rasin, J. H. Hayes, and M. Vierhauser. Cold-start software analytics. In *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*, pages 142–153, 2016. doi: 10.1145/2901739.2901740. URL http://doi.acm.org/10.1145/2901739.2901740.

61. J. Guo, J. Cheng, and J. Cleland-Huang. Semantically enhanced software traceability using deep learning techniques. In S. Uchitel, A. Orso, and M. P. Robillard, editors, *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*, pages 3–14. IEEE / ACM, 2017. ISBN 978-1-5386-3868-2. URL http://dl.acm.org/citation.cfm?id=3097370.

62. J. Guo, M. Gibiec, and J. Cleland-Huang. Tackling the term-mismatch problem in automated trace retrieval. *Empirical Software Engineering*, 22(3):1103–1142, 2017. ISSN 1573-7616. doi: 10.1007/s10664-016-9479-8. URL http://dx.doi.org/10.1007/s10664-016-9479-8.

63. S. Hayashi, T. Yoshikawa, and M. Saeki. Sentence-to-code traceability recovery with domain ontologies. In J. Han and T. D. Thu, editors, *APSEC*, pages 385–394. IEEE Computer Society, 2010. URL http://dblp.uni-trier.de/db/conf/apsec/apsec2010.html#HayashiYS10.

64. J. H. Hayes, A. Dekhtyar, S. K. Sundaram, and S. Howard. Helping analysts trace requirements: An objective look. In *RE '04: Proceedings of the Requirements Engineering Conference, 12th IEEE International*, pages 249–259, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2174-6. doi: http://dx.doi.org/10.1109/RE.2004.26.

65. J. H. Hayes, A. Dekhtyar, and S. K. Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Trans. Software Eng.*, 32(1): 4–19, 2006.

66. S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1994. ISBN 0023527617.

67. E. Hill, Z. P. Fry, H. Boyd, G. Sridhara, Y. Novikova, L. L. Pollock, and K. Vijay-Shanker. AMAP: automatically mining abbreviation expansions in programs to enhance software maintenance tools. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories, MSR 2008 (Co-located with ICSE), Leipzig, Germany, May 10-11, 2008, Proceedings*, pages 79–88, 2008. doi: 10.1145/1370750.1370771. URL http://doi.acm.org/10.1145/1370750.1370771.

68. F. Hill, K. Cho, and A. Korhonen. Learning distributed representations of sentences from unlabelled data. In *Proceedings of the 2016 Conference of the North American*

Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 1367–1377, San Diego, California, June 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N16-1162.

69. P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, editors. *OWL 2 Web Ontology Language: Primer*. W3C Recommendation, 27 October 2009. Available at http://www.w3.org/TR/owl2-primer/.

70. S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

71. J. Hu, G. Wang, F. Lochovsky, J.-t. Sun, and Z. Chen. Understanding user's query intent with wikipedia. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 471–480, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-487-4. doi: http://doi.acm.org/10.1145/1526709.1526773.

72. E. Hull, K. Jackson, and J. Dick. *Requirements engineering*. Springer Science & Business Media, 2010.

73. ISO. Iso/ts 21547:2010, health informatics security requirements for archiving of electronic health records. *International Organization for Standards*, TC 215 Health Informatics(http://www.iso.org/iso/home.htm (Last accessed 12/20/10)), 2010. doi: http://www.iso.org/iso/home.htm.

74. M. Iyyer, J. L. Boyd-Graber, L. M. B. Claudino, R. Socher, and H. Daumé III. A neural network for factoid question answering over paragraphs. In *EMNLP*, pages 633–644, 2014.

75. P. Jackson. *Introduction To Expert Systems (3 ed.)*. Addison Wesley, 1998.

76. A. Karpathy, J. Johnson, and F. Li. Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078, 2015. URL http://arxiv.org/abs/1506.02078.

77. R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler. Skip-thought vectors. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, NIPS'15, pages 3294–3302, Cambridge, MA, USA, 2015. MIT Press. URL http://dl.acm.org/citation.cfm?id=2969442.2969607.

78. D. Klein and C. D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 423–430, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. doi: 10.3115/1075096.1075150. URL http://dx.doi.org/10.3115/1075096.1075150.

79. J. C. Knight. Safety critical systems: challenges and directions. In *Proceedings of the 24th International Conference on Software Engineering, ICSE 2002, 19-25 May 2002, Orlando, Florida, USA*, pages 547–550, 2002. doi: 10.1145/581339.581406. URL http://doi.acm.org/10.1145/581339.581406.

80. L. Kof, R. Gacitua, M. Rouncefield, and P. Sawyer. Concept mapping as a means of requirements tracing. In *MaRK'10)*, 2010.

81. P. Kruchten. An ontology of architectural design decisions. *Groningen Workshop on Software Variability management*, pages 55–62, 2004.

82. A. N. Lam, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen. Combining deep learning with information retrieval to localize buggy files for bug reports (n). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 476–481. IEEE, 2015.

83. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

84. O. Levy, Y. Goldberg, and I. Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

85. Y. Li and J. Cleland-Huang. Ontology-based trace retrieval. In *Traceability in Emerging Forms of Software Engineering (TEFSE2013*, San Francisco, USA, May 2009.

86. D. Lin. An information-theoretic definition of similarity. In *ICML*, volume 98, pages 296–304, 1998.

87. J. Lin, C. C. Lin, J. Cleland-Huang, R. Settimi, J. Amaya, G. Bedford, B. Berenbach, O. B. Khadra, C. Duan, and X. Zou. Poirot: A distributed tool supporting enterprise-wide automated traceability. In *RE*, pages 356–357, 2006.

88. M. Lindvall and K. Sandahl. Practical implications of traceability. *Softw., Pract. Exper.*, 26(10):1161–1180, 1996.

89. S. Lohar, S. Amornborvornwong, A. Zisman, and J. Cleland-Huang. Improving trace accuracy through data-driven configuration and composition of tracing features. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013*, pages 378–388, 2013. doi: 10.1145/2491411.2491432. URL `http://doi.acm.org/10.1145/2491411.2491432`.

90. T. Long, R. Lowe, J. C. K. Cheung, and D. Precup. Leveraging lexical resources for learning entity embeddings in multi-relational data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 112–117, Berlin, Germany, August 2016. Association for Computational Linguistics. URL `http://anthology.aclweb.org/P16-2019`.

91. A. D. Lucia, R. Oliveto, and P. Sgueglia. Incremental approach and user feedbacks: a silver bullet for traceability recovery. *Software Maintenance, IEEE International Conference on*, 0:299–309, 2006. ISSN 1063-6773. doi: http://doi.ieeecomputersociety.org/10.1109/ICSM.2006.32.

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

93. A. D. Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk. Information retrieval methods for automated traceability recovery. In *Software and Systems Traceability.*, pages 71–98. Springer London, 2012. doi: 10.1007/978-1-4471-2239-5_4. URL http://dx.doi.org/10.1007/978-1-4471-2239-5_4.

94. P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang. Strategic traceability for safety-critical projects. *IEEE Software*, 30(3):58–66, 2013. doi: 10.1109/MS.2013.60. URL http://dx.doi.org/10.1109/MS.2013.60.

95. A. Mahmoud, N. Niu, and S. Xu. A semantic relatedness approach for traceability link recovery. In *IEEE 20th International Conference on Program Comprehension, ICPC 2012, Passau, Germany, June 11-13, 2012*, pages 183–192, 2012. doi: 10.1109/ICPC.2012.6240487. URL http://dx.doi.org/10.1109/ICPC.2012.6240487.

96. A. Marcus and J. I. Maletic. Using latent semantic analysis to identify similarities in source code to support program understanding. In *ICTAI '00: Proceedings of the 12th IEEE International Conference on Tools with Artificial Intelligence*, page 46, Washington, DC, USA, 2000. IEEE Computer Society.

97. A. Marcus and J. I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 125–135, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1877-X.

98. N. D. Matragkas, D. S. Kolovos, R. F. Paige, and A. Zolotas. A traceability-driven approach to model transformation testing. In *Proceedings of the Second Workshop on the Analysis of Model Transformations (AMT 2013), Miami, FL, USA, September 29, 2013*, 2013. URL http://ceur-ws.org/Vol-1077/amt13_submission_7.pdf.

99. J. Maxwell and A. Anton. Developing production rule models to aid in acquiring requirements from legal texts. In *Requirements Engineering Conference, 2009. RE '09. 17th IEEE International*, pages 101–110, 2009. doi: 10.1109/RE.2009.21.

100. T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

101. T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. URL http://arxiv.org/abs/1301.3781.

102. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013. URL http://arxiv.org/abs/1310.4546.

103. M. Mirakhorli and Jane Cleland-Huang. Detecting, tracing, and monitoring architectural tactics in code. *IEEE Trans. Software Eng.*, 42(3):205–220, 2016. doi: 10.1109/TSE.2015.2479217. URL http://dx.doi.org/10.1109/TSE.2015.2479217.

104. M. Mirakhorli, Y. Shin, J. Cleland-Huang, and M. Çinar. A tactic-centric approach for automating traceability of quality concerns. In *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, pages 639–649, 2012. doi: 10.1109/ICSE.2012.6227153. URL http://dx.doi.org/10.1109/ICSE.2012.6227153.

105. M. Mirakhorli, Y. Shin, J. Cleland-Huang, and M. Cinar. A tactic-centric approach for automating traceability of quality concerns. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 639–649, Piscataway, NJ, USA, 2012. IEEE Press. ISBN 978-1-4673-1067-3. URL http://dl.acm.org/citation.cfm?id=2337223.2337298.

106. L. G. P. Murta, A. van der Hoek, and C. M. L. Werner. Archtrace: Policy-based support for managing evolving architecture-to-implementation traceability links. In *ASE '06: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, pages 135–144, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2579-2. doi: http://dx.doi.org/10.1109/ASE.2006.16.

107. P. Mder, P. L. Jones, Y. Zhang, and J. Cleland-Huang. Strategic traceability for safety-critical projects. *IEEE Software*, 30(3):58–66, May 2013. ISSN 0740-7459. doi: 10.1109/MS.2013.60.

108. M. Nauman and S. Khan. Using personalizedweb search for enhancing common sense and folksonomy based intelligent search systems. In *WI '07: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, pages 423–426, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-3026-5. doi: http://dx.doi.org/10.1109/WI.2007.108.

109. A. D. Nicola, M. Missikoff, and R. Navigli. A software engineering approach to ontology building. *Inf. Syst.*, 34(2):258–275, 2009.

110. A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 522–531, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-3076-3. URL http://dl.acm.org/citation.cfm?id=2486788.2486857.

111. T. Parsons. Thematic relations and arguments. *Linguistic Inquiry*, 26(4):635–662, 1995.

112. R. Pascanu, Ç. Gülçehre, K. Cho, and Y. Bengio. How to construct deep recurrent neural networks. *CoRR*, abs/1312.6026, 2013. URL http://arxiv.org/abs/1312.6026.

113. PCI. Pci security standard quick reference guide. *Payment Card Industry Security Guidelines*, https://www.pcisecuritystandards.org (Last accessed 12/30/10), 2006. doi: https://www.pcisecuritystandards.org.

114. J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

115. N. Pinto, D. Doukhan, J. J. DiCarlo, and D. D. Cox. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS Comput Biol*, 5(11):e1000579, 2009.

116. M. Porter. Porter's stemming algorithm. In *Program*, pages 130–137, 1980.

117. B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *IEEE Trans. Softw. Eng.*, 27:58–93, January 2001. ISSN 0098-5589. doi: 10.1109/32.895989. URL http://portal.acm.org/citation.cfm?id=359555.359578.

118. B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *IEEE transactions on software engineering*, 27(1):58–93, 2001.

119. V. Raychev, M. Vechev, and E. Yahav. Code completion with statistical language models. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, pages 419–428, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2784-8. doi: 10.1145/2594291.2594321. URL http://doi.acm.org/10.1145/2594291.2594321.

120. T. Rocktäschel, E. Grefenstette, K. M. Hermann, T. Kociský, and P. Blunsom. Reasoning about entailment with neural attention. *CoRR*, abs/1509.06664, 2015. URL http://arxiv.org/abs/1509.06664.

121. *DO-178C: Software Considerations in Airborne Systems and Equipment Certification*. RTCA and EUROCAE, standard document no. do-178c/ed-12c edition, 2012.

122. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

123. G. Salton. *Automatic text processing: the transformation, analysis, and retrieval of information by computer*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989. ISBN 0-201-12227-8.

124. G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, NY, USA, 1986.

125. G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975. doi: 10.1145/361219.361220. URL http://doi.acm.org/10.1145/361219.361220.

126. Sarbanes-Oxley. A guide to the sarbanes-oxley act. *The Sarbanes-Oxley Compliance Guide*, 2002(http://www.soxtoolkit.com/ (Last accessed on 12/30/10)), 2002.

127. J. Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.

128. M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

129. D. Shen, J.-T. Sun, Q. Yang, and Z. Chen. Building bridges for web query classification. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 131–138, New York, NY, USA, 2006. ACM. ISBN 1-59593-369-7. doi: http://doi.acm.org/10.1145/1148170.1148196.

130. Y. Shin and J. Cleland-Huang. A comparative evaluation of two user feedback techniques for requirements trace retrieval. In *SAC*, pages 1069–1074, 2012.

131. Y. Shin, J. H. Hayes, and J. Cleland-Huang. Guidelines for benchmarking automated software traceability techniques. In *8th IEEE/ACM International Symposium on Software and Systems Traceability, SST 2015, Florence, Italy, May 17, 2015*, pages 61–67, 2015. doi: 10.1109/SST.2015.13. URL http://dx.doi.org/10.1109/SST.2015.13.

132. P. Shvaiko and J. Euzenat. Ontology matching: State of the art and future challenges. *IEEE Trans. Knowl. Data Eng.*, 25(1):158–176, 2013.

133. R. Socher, C. C. Lin, C. Manning, and A. Y. Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.

134. G. Spanoudakis, A. Zisman, E. Pérez-Miñana, and P. Krause. Rule-based generation of requirements traceability relations. *Journal of Systems and Software*, 72(2):105–127, 2004.

135. Stanford. Protégé: Open source ontology editor, 2013. URL http://protege.stanford.edu/.

136. H. Sultanov and J. H. Hayes. Application of reinforcement learning to requirements engineering: requirements tracing. In *21st IEEE International Requirements Engineering Conference, RE 2013, Rio de Janeiro-RJ, Brazil, July 15-19, 2013*, pages 52–61, 2013. doi: 10.1109/RE.2013.6636705. URL http://dx.doi.org/10.1109/RE.2013.6636705.

137. H. Sultanov, J. H. Hayes, and W. Kong. Application of swarm techniques to requirements tracing. *Requir. Eng.*, 16(3):209–226, 2011. doi: 10.1007/s00766-011-0121-4. URL http://dx.doi.org/10.1007/s00766-011-0121-4.

138. K. S. Tai, R. Socher, and C. D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

139. A. Tang, Y. Jin, and J. Han. A rationale-based architecture model for design traceability and reasoning. *Journal of Systems and Software*, 80(6):918 – 934, 2007. ISSN 0164-1212. doi: DOI:10.1016/j.jss.2006.08.040. URL http://www.sciencedirect.com/science/article/B6V0N-4M6SB7T-1/2/82ea7eabc2c4947aee2168fd536cc9f3.

140. T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2), 2012.

141. D. Tufis and O. Mason. Tagging romanian texts: a case study for qtag, a language independent probabilistic tagger. In *Proceedings of the First International Conference on Language Resources and Evaluation (LREC*, pages 589–596, 1998.

142. S. Wang, T. Liu, and L. Tan. Automatically learning semantic features for defect prediction. In *Proceedings of the 38th International Conference on Software Engineering*, pages 297–308. ACM, 2016.

143. C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000. ISBN 0-7923-8682-5.

144. World Vista. Department of Veterans Affairs Office of Enterprise Development vistahealthvet monograph.

145. B. Yurekli, G. Capan, B. Yilmazel, and O. Yilmazel. Guided navigation using query log mining through query expansion. In *NSS '09: Proceedings of the 2009 Third International Conference on Network and System Security*, pages 560–564, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3838-9. doi: http://dx.doi.org/10.1109/NSS.2009.27.

146. Y. Zhang, R. Witte, J. Rilling, and V. Haarslev. Ontological approach for the semantic recovery of traceability links between software artefacts. *Software, IET*, 2(3):185 – 203, june 2008. ISSN 1751-8806. doi: 10.1049/iet-sen:20070062.

147. X. Zou, R. Settimi, and J. Cleland-Huang. Evaluating the use of project glossaries in automated trace retrieval. In H. R. Arabnia and H. Reza, editors, *Software Engineering Research & Practice, SERP 2008, Las Vegas Nevada, USA*, pages 157–163. CSREA Press, 2008. ISBN 1-60132-088-4.

148. X. Zou, R. Settimi, and J. Cleland-Huang. Improving automated requirements trace retrieval: a study of term-based enhancement methods. *Empirical Software Engineering*, 15(2):119–146, 2010.