Improving Trace Accuracy through Data-Driven Configuration and Composition of Tracing Features

Sugandha Lohar, Sorawit Amornborvornwong
DePaul University
Chicago, IL, USA
sonul.123@gmail.com,
sorambww@hotmail.com

Andrea Zisman
Department of Computing
The Open University
Milton Keynes, MK7 6AA, UK
andrea.zisman@open.ac.uk

Jane Cleland-Huang
DePaul University
Systems and Requirements Engineering Center
Chicago, IL, USA
jhuang@cs.depaul.edu

ABSTRACT

Software traceability is a sought-after, yet often elusive quality in large software-intensive systems primarily because the cost and effort of tracing can be overwhelming. State-of-the art solutions address this problem through utilizing trace retrieval techniques to automate the process of creating and maintaining trace links. However, there is no simple one-size-fits all solution to trace retrieval. As this paper will show, finding the right combination of tracing techniques can lead to significant improvements in the quality of generated links. We present a novel approach to trace retrieval in which the underlying infrastructure is configured at runtime to optimize trace quality. We utilize a machine-learning approach to search for the best configuration given an initial training set of validated trace links, a set of available tracing techniques specified in a feature model, and an architecture capable of instantiating all valid configurations of features. We evaluate our approach through a series of experiments using project data from the transportation, healthcare, and space exploration domains, and discuss its implementation in an industrial environment. Finally, we show how our approach can create a robust baseline against which new tracing techniques can be evaluated.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Life cycle*

General Terms

Documentation, Management

Keywords

Trace retrieval, configuration, trace configuration

1. INTRODUCTION

Software traceability is an important element of the development process, especially in large, complex, or safety-critical software-intensive systems [4]. It is used to capture relationships between requirements, design, code, test-cases, and other software engineering artifacts, and support critical activities such as impact analysis, compliance verification, test-regression selection, and safety-analysis. As such, traceability is mandated in safety-critical domains including the automotive, aeronautics, and medical device industries. Unfortunately, tracing costs can grow excessively high if trace links have to be created and maintained manually by human users, and as a result, practitioners often fail to establish adequate traceability in a project [27].

To address these needs, numerous researchers have developed or adopted algorithms that semi-automate the process of creating trace links. These algorithms include the Vector Space Model (VSM) [23], Probabilistic approaches [14], Latent Semantic Indexing [2, 12], Latent Dirichlet Allocation (LDA) [13], rule-based approaches that identify relationships across project artifacts [35], and approaches that identify artifacts committed as part of the same change to a version control systems [21] or modified consecutively by a single user [3].

Given such a profusion of traceability techniques, and the multiple ways in which each technique can be configured or combined with others, it is difficult to know which combination of techniques to use for a specific dataset and/or project. The problem of finding the right configuration is particularly pertinent as researchers have not yet been able to provide clear guidelines as to which tracing techniques are most effective on different kinds of datasets. As the results reported in this paper will show, finding the right configuration can lead to very significant improvements in the accuracy of generated trace links, in some cases improving trace accuracy by over 100%. This is particularly notable because there is currently no individual tracing technique that has been shown to consistently outperform other techniques across all datasets. Industrial adoption of trace generation methods will be better supported if best-of-breed techniques can be discovered and used within the context of a specific project. Furthermore, research advances will be facilitated if new techniques can be compared against the best combination of existing techniques instead of against a single baseline technique which performs inconsistently across different datasets.

In this paper we therefore present a state-of-the-art approach for configuring a traceability infrastructure in order to improve the achieved accuracy of the generated trace links. We define traceability infrastructure as the enterprise-

ESEC/FSE'13, August 18–26, 2013, Saint Petersburg, Russia
ACM 978-1-4503-2237-9/13/08
http://dx.doi.org/10.1145/2491411.2491432

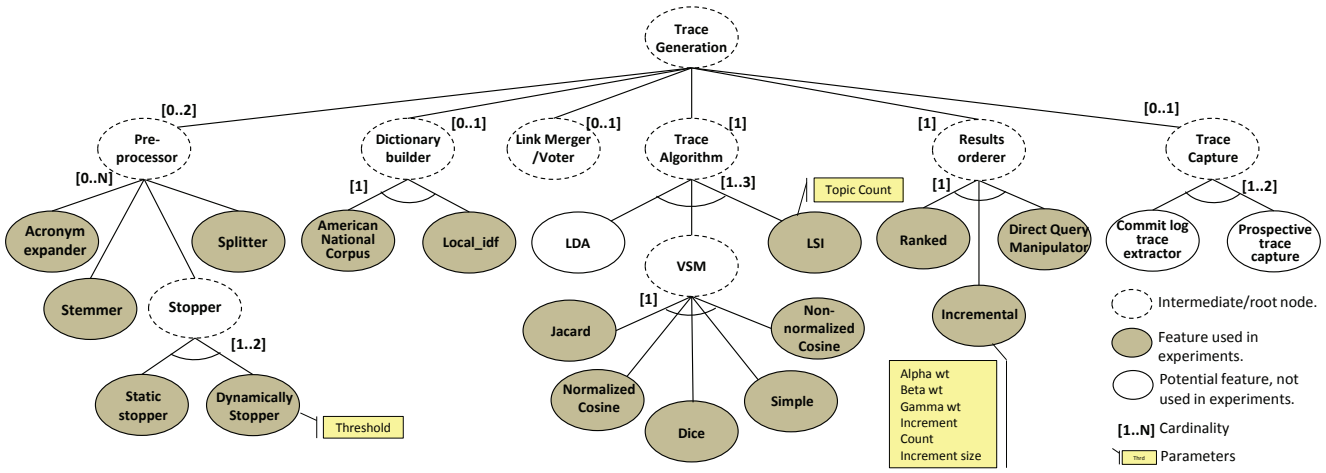


Figure 1: A Selection of trace algorithms and utilities represented in a feature model.

wide instrumentation needed to support all aspects of the traceability process, including GUI (graphical user interface) components, plug-ins for case tools, files for storing traceability data, and also trace-generation and traceability management tools. Our approach, which we refer to as *Dynamic Trace Configuration* (DTC), utilizes an initial training set of validated trace links to discover a *good* configuration of tracing techniques. We define a good configuration as one that delivers high accuracy of the generated trace links. Furthermore, as our approach does not guarantee to find the optimal approach, we also introduce the concept of *top* configuration as the best configuration found by our DTC approach for a given dataset and feature model. Given the potentially large combination of techniques, and the slow running time of some of the tracing algorithms, we utilize a Genetic Algorithm (GA) to intelligently search through the space of viable configurations in order to find the best performing configuration. We justify our choice of the GA in Section 2.3. While other researchers have explored different techniques for combining predefined sets of tracing techniques [6, 13, 18, 19], the novel contribution of our work is the dynamic and scalable approach for learning what can be seen as a good composition and configuration of a very broad and potentially expanding set of tracing techniques.

The remainder of the paper is structured as follows. Section 2 describes the DTC infrastructure and processes including the use of feature modeling, simulation, and intelligent search. Section 3 describes a series of experiments that we conducted to evaluate the efficacy of our approach for finding an effective configuration, and for using the identified configuration in an evolving software project. Section 4 discusses the practical applications in industrial projects and research domains. Section 5 discusses threats to validity and Section 6 describes related research from areas of traceability and self-adaptation environments. Finally, Section 7 summarizes our findings and discusses future work.

2. DYNAMIC TRACE CONFIGURATION

Our approach takes as input a training set of source and target artifacts and a matrix of human validated trace links which serve as a "reference set" for improving the accuracy of the generated trace links. It also imports a feature model of available tracing techniques, and then searches for the combination and configuration of features that increase the accuracy of automatically generated trace links when applied to the reference set. The DTC includes three primary elements: (1) a feature model specifying the available trace algorithms and utilities (from now on referred to as 'features'), (2) a simulation environment in which any viable configuration of features in the feature model can be instantiated and then used to generate trace links against a given dataset, and (3) an intelligent search component capable of directing a search through the space of all viable configurations in order to find a good configuration for a given dataset. In the following sections we describe each of these in more detail.

2.1 Modeling Features and Configurations

DTC utilizes a feature model to specify the set of available features and their constraints [8]. While alternate representations are possible [18], feature models have been shown to be expressive and scalable across many different domains, and can therefore support a wide variety of features and interactions. Utilizing a feature model not only accommodates the features of the current experiment, but can easily scale up to incorporate new features as they are proposed by researchers and become available to practitioners. Furthermore, given the widespread interest in feature modeling, several different tools are available for generating valid configurations and/or checking the correctness of candidate configurations.

In this work we represent the feature model using the Textual Variability Language (TVL) [8] and associated tools developed by Heymans et al. TVL is a text-based feature modeling language that provides a rich syntax and formal semantics. The TVL tool provides a syntax checker which was used to initially validate the well-formedness of the trace feature model. It also provides support for querying whether a specific configuration is valid. For illustrative purposes we depict a small and representative set of popular tracing techniques in Figure 1 using a graphical display of a feature model.

This feature model contains several basic features. Preprocessors are used to prepare raw data for tracing. For ex-

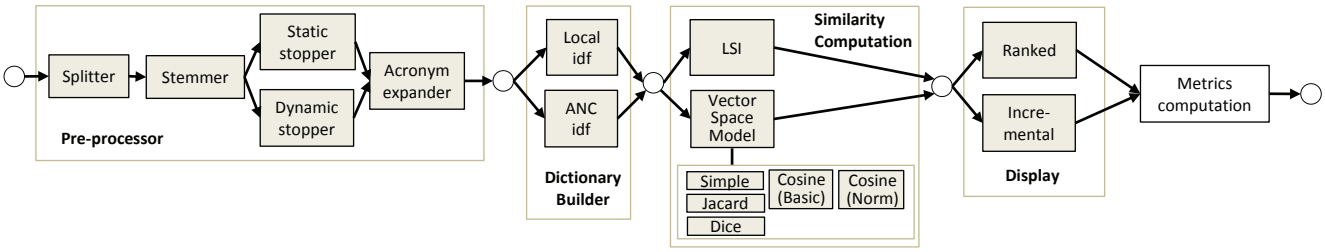


Figure 2: Pipe-and-filter Architecture used to instantiate valid combinations of tracing features.

ample, a stopper removes commonly occurring terms such as "this" and "that", which are not useful for tracing purposes, while an acronym expander expands acronyms such as "RBAC" to their extended forms (in this case "Role based access control"). The dictionary builder constructs an index of terms found in various documents, and computes term weightings using algorithms such as tf-idf (term frequency - inverse document frequency) based on terms found in the project documents, or on terms found in the American National Corpus. Core algorithms such as VSM (Vector Space Model), LSI (Latent Semantic Indexing) and LDA (Latent Dirichlet Analysis) compute the similarity between pairs of documents. Latent Semantic Indexing (LSI) [2, 12, 16] utilizes a mathematical technique called Singular Value Decomposition (SVD) to uncover the underlying latent semantic structure of word usage in unstructured text, and then uses these latent topics to match queries and documents that are conceptually similar in meaning regardless of whether they share specific words. The Vector Space Model (VSM) [23] represents queries and documents as vectors of terms, and then applies the standard term-frequency inverse-document frequency (tf-idf) approach to compute similarity scores between artifacts based upon the frequency at which the term occurs in both the source and target artifact and the inverse frequency of its occurrence across the collection of documents.

Generated trace links can be presented to the user in several different ways, including a basic ranked approach in which links are presented in the order of the generated similarity scores, an incremental approach in which links are presented incrementally to the user and relevance feedback is used to reorder the remaining links, and finally the Direct Query Manipulation (DQM) approach, in which a user modifies the query in order to filter out unwanted results [32]. Finally, components such as commit parsers [21], rule-based link generation techniques [35], and tool-based monitors [3] can also be used to generate trace links.

A feature model also specifies the cardinality of features and depicts whether a given feature is mandatory or optional. In addition to the visual representation shown in Figure 1, additional *requires* and *constrains* relationships must also be specified. Examples include: *"VSM requires dictionary builder"* or *"[trace algorithm>1] requires Link Merger/Voter"*. While most features are binary in nature (i.e. either present or not present), some features such as LSI must be configured prior to use. The feature model documents the range of allowed values for each parameter.

2.2 Evaluating a Configuration

Our approach requires various candidate configurations to be evaluated by utilizing the configuration to generate trace links from source to target artifacts. The resulting trace links are then compared against the reference trace matrix to determine how well the configuration performed. The metric of Mean Average Precision (MAP) is used for evaluative purposes. MAP computes the extent to which correct links are placed at the top of the ranked list of generated trace links. Because our implementation of MAP examines all correct links it also assumes recall (i.e. the ability to retrieve correct links) of 100%. The use of MAP as a traceability measure has been advocated in numerous papers [36, 38]. First the average precision (AP) of each query is computed:

$$AP = \frac{\sum_{r=1}^{N}(Precision(r) \times isRelevant(r))}{|RelevantDocuments|} \quad (1)$$

where r is the rank of the target artifact in an ordered list of links, $isRelevant()$ is a binary function assigned 1 if the link is relevant (i.e. marked as correct in the reference set) and 0 otherwise, $P(r)$ is the precision computed after truncating the list immediately below that ranked position, and N is the total number of documents. When multiple links are listed for a single similarity score (e.g. at similarity of zero) the links are evenly distributed across the space of that score, simulating their random distribution. MAP is then computed across all queries as follows:

$$MAP = \frac{\sum_{q=1}^{Q} AP(q)}{Q} \quad (2)$$

where, q is a single query and Q is the total number of queries.

To determine the efficacy of a configuration for tracing a particular dataset it is necessary to provide an architecture that can accommodate all valid combinations of available features. For purposes of the experiments described in this paper we designed the architecture, shown in Figure 2, to accommodate all of the features shaded in Figure 1. This architecture utilizes a pipe-and-filter pattern and assumes a fairly rigid sequencing of processes. Components in the pipeline can be turned on or off, depending on whether they are required in the configuration. The architecture provides one possible solution for configuring any valid configuration of features from our feature model. However, alternate, and more flexible architectural designs are also possible. For example, a more extensive architecture could accommodate the dynamic sequencing of preprocessors and/or allow results from multiple tracing techniques to be merged through voting and or other combinatory techniques.

2.3 Searching for the Best Configuration

Utilizing a brute force approach to evaluate every single candidate configuration can be infeasible given that the fea-

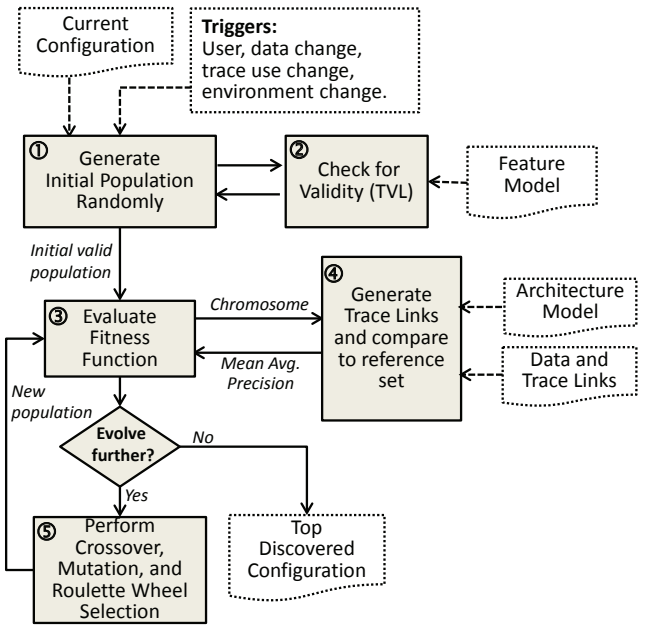


Figure 3: The overall DTC Process

ture model shown in Figure 1 includes 1,644,408 valid configurations without even considering parameterizations of the individual components. Assuming reasonable increments for each parameter (i.e. increments of 5 for values ranging from 0 (or 5) to 500, increments of 1 for values ranging from 1-20, and increments of 0.05 for values ranging from 0.0-1.0) the number of valid configurations jumps to 7.033383E+12. In our experiments we observed that for large datasets and candidate configurations that include slow running algorithms such as LSI, it takes approximately two hours to generate a complete set of trace links from source to target artifacts. We estimate the average time over all candidate configurations for our largest data set (See Section 3) to be 40 minutes per configuration. Therefore, generating traces for every configuration would take approximately five hundred and thirty-five million years to run. This is clearly infeasible and therefore a more intelligent approach is needed for searching through the space of viable configurations.

There are several viable searching techniques; however we chose to utilize a Genetic Algorithm (GA) [20] because it is a natural fit for our problem and is relatively straightforward to implement. A GA mimics the evolutionary process found in nature. It involves the steps of (i) modeling a candidate solution as a chromosome, (ii) creating an initial population of chromosomes, (iii) computing fitness functions to evaluate the efficacy of a given configuration, (iv) evolving the population to the next generation, and (v) selecting the best known configuration. Figure 3 shows an overview of the main GA steps. These steps are described below.

2.3.1 Modeling Candidate Solutions

A GA fits our problem because each candidate trace configuration can easily be modeled as a chromosome encoded as a string of bits, where each bit represents a different feature that is either present (1) or not present (0) in the configuration. As shown in Figure 4, configurable parameters are represented as sub-chromosomes. For experimental purposes we included 22 features with four configurable ones. For example, the incremental ranking component is configured by five parameters (alpha, beta, gamma, increment count, and increment size), all of which are represented in the sub-chromosome.

2.3.2 Creating an Initial Population

For each run of the GA, an initial population of 50 chromosomes is created (See Step 1 in Figure 3). As previously explained, the feature model is specified using the Textual Variability Language (TVL) [8]. Chromosomes are passed as queries to the TVL tool to check their validity (See Step 2). Invalid configurations are rejected, and the process is continued until the complete population is built.

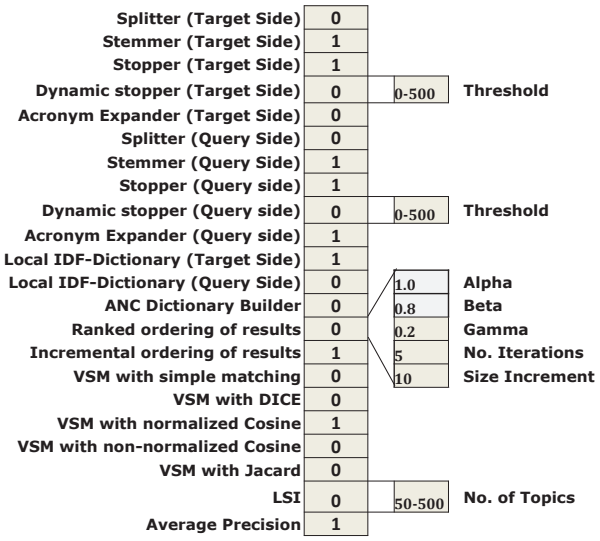


Figure 4: The Genetic Algorithm represents tracing configurations as a chromosome.

2.3.3 The Fitness Function

The fitness function of each chromosome is computed by generating a trace configuration according to the features specified in the chromosome, and then using this configuration to generate trace links for the training set (See Steps 3 and 4 in Figure 3). The MAP score, which is computed by comparing the generated links to the reference matrix for each dataset, serves as the fitness function for the GA algorithm.

2.3.4 Evolving the Population

A stochastic process is used to select the best chromosomes to be carried forward as parents into the next generation. Our implementation carries ten chromosomes forward. One of these is the *elite* chromosome, i.e. the chromosome with the highest MAP value from the previous generation. The remaining nine chromosomes are selected using a standard practice based on the *roulette wheel*, which works on the premise that chromosomes scoring higher MAP values have a greater chance of survival than weaker ones. Basically, fitter individuals are given proportionally higher chances of being selected during a spin of the roulette wheel than less fit ones. In our implementation we select only ten parents

based on an initial series of experiments designed to fine-tune performance by minimizing runtime and maximizing accuracy of the GA algorithm.

GAs typically use two techniques of *cross-over* and *mutation* to generate offspring chromosomes from the parents in order to create the new generation (Step 5). These two practices emulate the natural process of carrying forward genetic material from the parents while introducing sufficient variation to potentially surpass their parents in terms of fitness. A *mutation* involves flipping one of the genomes (bits) in the chromosome, i.e. flipping a 1 to 0 in order to remove a feature, or flipping a 0 to 1 in order to add a feature. Based on initial experimentation, a mutation rate of 0.25 was established for our experiments. To execute a mutation, a number N from 1-5 is randomly selected to represent the number of bits to be flipped. N unique bits are then randomly selected and flipped. If a sub-chromosome exists for any of the selected bits, one of the sub-chromosome's bits are randomly switched to a score within the range of allowed values. A *cross-over* involves creating two children from the genetic material of two parent chromosomes. Cross-overs are executed by selecting two parents, choosing a random position in the chromosome, using this position to partition each of the chromosomes into head and tail sections, and then exchanging the two tails.

2.3.5 Computing Fitness Functions

For purposes of our experiments we established a stopping condition that halts the GA if no improvements are found after 5 successive generations, and otherwise halts after the 60th generation. Once the experiment halts, the highest performing configuration across all generations is selected. The visualization tool depicted in Figure 5 was developed to support experimentation by displaying the MAP score of configurations discovered through multiple generations of the GA search. It depicts results from the first seven generations for a healthcare dataset (i-Trust) described in Section 3. As a sanity check we compared the output of our GA to previously published results which were available for some of the datasets. These results confirmed that our approach consistently produced either improved or comparative MAP scores [33].

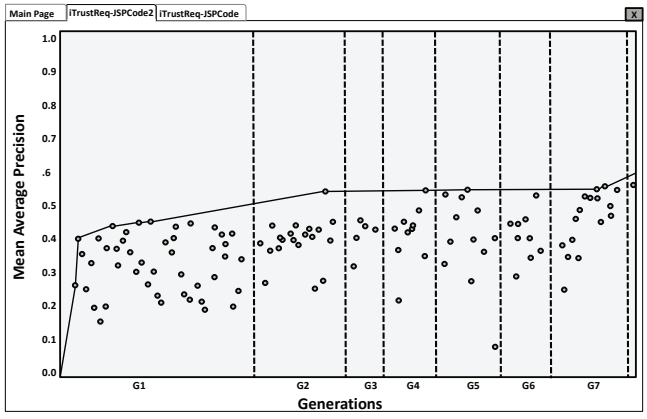


Figure 5: Visualization of the growth in MAP scores (y-axis) over progressive generations (x-axis) of the GA search process

3. EVALUATING DTC CUSTOMIZATION

While the goal of DTC is to find the optimal trace configuration for a dataset given an available set of tracing features, a GA algorithm does not guarantee to do so. For any form of dynamic adaptation to be worthwhile in a system, the benefits gained from changing the system must outweigh the associated costs. In this section of the paper we evaluate the benefits of customizing the trace infrastructure for a specific project. In the following section we explore the practicality and processes that might be followed in order to integrate reconfiguration into a real project environment.

3.1 Experimental Design

DTC was evaluated against six different projects from the transportation, healthcare, and space domains as depicted in Table 1. In the table, the number of the source and target elements are represented in brackets. For example, Industry 1 has 442 unique requirements and 3,104 unique design artifacts. Four of our datasets were obtained from industry or government sources, while the remaining two datasets represent academic projects. The two transportation sector projects were provided under non-disclosure agreements and are, therefore, simply referred to as Industry-1 and Industry-2. All datasets except for Industry 1 and 2 are publicly available through CoEST.org. For each dataset, the confirmed trace matrix was provided either by the industrial source, or else previously used and validated by several different research groups [2, 23, 33].

Table 1: Datasets used in Study

Project	Source	Target	Links
Industry 1 (Transport)	Reqs (442)	Design (3104)	6961
Industry 2 (Transport)	Reqs (224)	Design (945)	700
I-Trust (Health)	Reqs (131)	Code (332)	535
CCHIT (Health)	Reg. Codes (453)	Reqs(958)	534
E-Clinic (Health)	Use Cases (30)	Test Cases (47)	63
CM-1 (NASA)	Reqs (22)	Design (46)	46

All of the experiments were conducted utilizing the subset of features shaded gray in Figure 1. Features were limited to those which were either already available to us as executable components, or which we were able to implement within the timeframe of this project. The architecture shown in Figure 2 was utilized as a framework capable of executing any valid combination of features. The experimental tracing environment was constructed using the TraceLab tool [15]. TraceLab is a highly flexible research platform that provides an extensible library of components for importing and preprocessing artifacts, for generating trace links using multiple techniques, and for reviewing and evaluating results. These components can be integrated into complex, executable workflows using the TraceLab plug and play environment. Data is exchanged between components via standard TraceLab datatypes using a blackboard architectural style. Our experimental test harness, which was created in TraceLab, was responsible for importing the data to be tested and then utilizing the DTC process to search for the best trace configuration.

3.2 Experiments

Experiments 1-3 were conducted to address three specific research questions (RQ) designed to evaluate the benefits and efficacy of dynamic trace configuration.

Table 2: Experiment 1 demonstrated that each dataset had a unique top configuration

Features	E-Clin C1	CM-1 C2	CCHIT C3	i-Trust C4	Ind-1 C5	Ind-2 C6
Achieved MAP	0.835	0.726	0.395	0.376	0.285	0.363
Stem	•	•	•	•	•	•
SourceStandardStop		•				
TargetStandardStop						
SourceDynStop	•		•	•	•	
TargetDynStop					•	
SplitSource	•	•	•	•		•
SplitTarget			•			•
SourceAcronym						
TargetAcronym		•		•		
tf-idfDictBld	•	•	•	•		•
ANCDictBld					•	
VectorSpModel	•	•		•	•	•
LSI			•			
Cosine(basic)	•	•		•		•
Cosine(norm)						
Dice						
Jacard						
SimpleMatch			•			
Incr.Display	•	•				
Rank.Display			•	•	•	•

SourceDynStop	27		46	43-49	6	
TargetDynStop					25	
RocchioIteration	10	9				
RocchioTopN	2	1				
RocchioGamma	0.316	0.829				
RocchioAlpha	0.464	0.829				
RocchioBeta	0.996	0.932				
LSI-k			320			

RQ1: Does each dataset of source and target artifacts have a distinct *top* performing trace configuration? The first research question investigated whether each of the six datasets depicted in Table 1 had a unique top configuration. We utilized DTC to discover the top configuration for each of the datasets. Running time of DTC varied from 10 minutes for smaller datasets (e.g., E-Clinic) to over 12 hours for larger datasets (e.g., Industry-1).

Results are reported in Table 2 and show that different datasets performed best with very different configurations. For each data set we report the achieved MAP score for the highest performing configuration (C_i), marked with a bullet (•) in the corresponding cells of the features in the configuration. The lower part of Table 2 shows the parameter values of the features in the respective configurations. For example the E-Clinic dataset returned a healthy MAP score of approximately 0.835 when using the VSM with local IDF. In contrast, the CCHIT data set performed best using LSI, achieving MAP scores of 0.395. Finally, the Industry-1 achieved its highest MAP scores of 0.285 using VSM combined with the ANC Dictionary builder. These results clearly show that different datasets have their own top scoring configurations. As a general observation, it should be noted that larger datasets tend to produce lower MAP scores due to the higher potential for false positive links.

Another interesting observation is that the results were not easily predictable in advance. In fact general wisdom suggests that LSI outperforms VSM on larger datasets; however in this experiment, LSI was not selected for the largest dataset i.e. Industry-1. Furthermore, the use of the global tf-idf, which had previously been discredited [9], led to very significant improvements in the Industry-1 dataset. These results highlight the value of using DTC to experimentally discover the optimal configuration for a given dataset instead of just adopting a default configuration.

RQ2: Is there a single configuration which performs well on all datasets? The previous experiment identified the *top* configuration for each dataset; however in a second experiment we explored the closely related question of whether any of these top configurations, would perform "well" across all six datasets. The underlying premise here is that if we could identify a single configuration which performed well for all datasets, then we should adopt that configuration as the default and avoid the costs associated with dynamic discovery and adaptation of a trace configuration.

In order to investigate these differences we used the *top* configuration achieved for each of the six datasets in Experiment 1 (labeled configurations C1 to C6 in Table 2), and applied each configuration against all of the other five datasets to obtain their respective MAP values. The results of this experiment are reported in Figure 6, and show that none of the tested configurations performed well across all datasets. The best overall configuration was $C4$ with a mean MAP of 0.423 over all six datasets. However, this configuration performed particularly poorly on the Industry-1 dataset, and was outperformed by three other techniques for E-Clinic.

These results, and the others reported in Figure 6 clearly show that no single configuration performed well across all datasets. We hypothesize that individual characteristics of each dataset influence the performance of the various tracing features. For this reason the "one size fits all" approach which has been the defacto standard until now does not appear to be effective.

RQ3: Does each pair of artifact types in a project have a distinct top performing configuration? In this experiment we evaluated whether the same configuration could be used to trace between different types of artifacts in the same project i.e. between requirements and regulatory standards, or between test cases and requirements. Easy Clinic, Industry-2, and I-Trust datasets were used for this experiment as each of these had multiple trace matrices as depicted in Table 3. DTC was run against each pair of artifact types for which a confirmed trace matrix was available. The experiment followed the same design used for Experiment 1. For each pair of artifacts in a project we used the top configuration achieved for that pair and applied it against all other pairs of artifacts in the same project to obtain their respective MAP values. The results are reported in Table 3, and show that each pair of source/target artifacts performed best when using its own customized configuration, meaning that a configuration identified for one pair of artifacts in a project should not necessarily be used for all other pairs of artifacts in a project. These results were not entirely unexpected given that different artifact types (i.e. requirements vs. code vs. test cases) exhibit very different characteristics in terms of document length and vocabulary used.

3.3 Analysis of Results

An analysis was performed to determine statistically if the use of the customized trace configurations was effective. Based on the results of Experiment 1 and Experiment 3, we identified the top configuration for each pair of source-target artifacts across all of the datasets. We refer to these configurations as *customized trace configurations*. For example

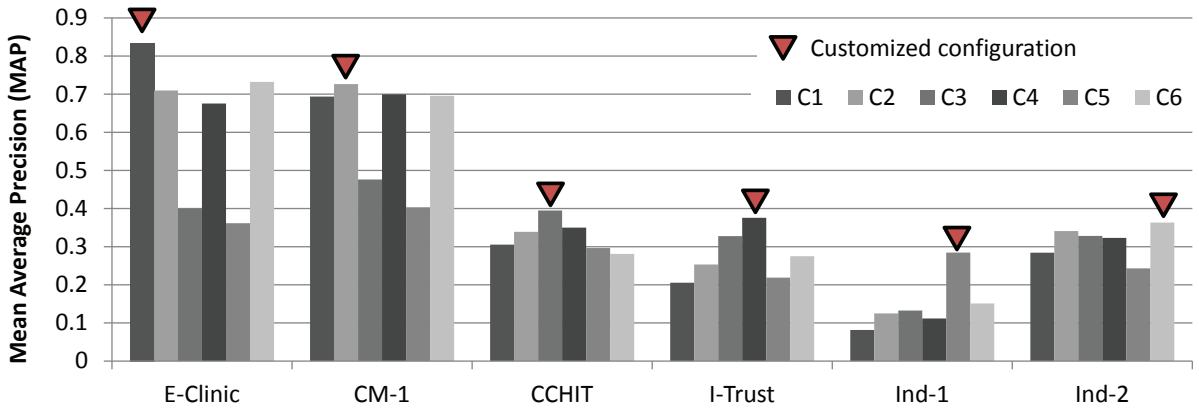


Figure 6: Mean Average Precision (MAP) obtained using the six winning configurations across all datasets

the customized trace configuration for Easy-Clinic dataset from Table 2 is $C1$, while all other tested configurations are considered *non-customized* configurations for that dataset.

Based on these definitions, a paired T-test was performed to determine if the use of customized trace configuration is effective. In this analysis, each customized configuration is paired with its relevant non-customized configuration. For example, in the Easy-Clinic dataset C1 is paired with its relevant C2, C3, C4, C5 and C6 (see Table 2) configurations, as well as C1.2 and C1.3 (see Table 3). (Please note that C1.1 is excluded because it is the same as C1). For 11 different pairs of source-target artifacts across six different datasets, there was 44 pair of *customized*:*non-customized* configuration pairs. We point out that if we could have performed pairwise comparisons against a far larger sample of configurations generated by the GA or randomly selected ones. We chose not to do this because we wanted our comparison to be against strongly viable configurations and not very low-performing configurations which would be unlikely to be used.

A mean improvement in MAP score of 0.14 (on a scale of 0.00-1.00) was observed for customized configurations, with a standard deviation of 0.113. This was significantly greater than zero (i.e. t (43) =2.01, one-tail p= 8.20E-11, and confidence interval of 95%) providing evidence that customized configurations are effective for improving trace accuracy. The 0.14 increase in MAP score represented a 49.45% improvement realized through using customized versus non customized configurations. Furthermore, given concerns over the normal distribution of the data, a Wilcoxon-Ranked Sum test was also conducted. This test rejected the null hypothesis at z-value=-5.7767 (with alpha = 0.01 and p<0.01), indicating that there is a significant difference between the two samples.

We also compared the raw distribution of MAP scores for customized versus non-customized configurations and plotted the results in the box and whisker graph shown in Figure 7. This graph provides an intuitive visualization of the difference in MAP scores between customized and non-customized configurations. For example, it shows that mean MAP scores for customized trace infrastructures is 0.54 versus 0.39 for non-customized configurations. Based on these results and the datasets included in our study, we conclude that customizing trace configurations significantly improves the quality of generated trace links across **all** datasets.

Table 3: Different configurations are needed for different traceability paths within the same project

Easy-Clinic	UC-TC (C1.1)	TC-Code (C1.2)	UC-Code (C1.3)
Use Case - Test Case	**0.8347**	0.5213	0.72
Test Case - Code	0.7025	**0.8616**	0.8186
Use Case - Code	0.4201	0.5944	**0.7978**

i-Trust	Req-Code (C2.1)	Req-JspCode (C2.2)	Req-JavaCode (C2.3)
Requirements - Code	**0.3756**	0.2846	0.3195
Requiremnts - JSP	0.4306	**0.6646**	0.3195
Requirements - Java	0.389	0.323	**0.4359**

Industry-2	SSRS-SDD (C3.1)	SDD-SRS (C3.2)
System Reqs - Design	**0.3632**	0.3269
Design - Software Reqs.	0.3654	**0.4061**

4. DTC IN PRACTICE

One of the primary objectives of DTC is to support ongoing customization of the traceability infrastructure in an industrial project in order to improve the effectiveness of trace retrieval methods in practice. In this phase of the work we experimentally investigated the use of DTC in an industrial setting by simulating the impact of a growing dataset on the trace configuration.

4.1 Experiments

Two additional experiments were conducted. The first explored the degree to which configurations were stable over time, while the second addressed the critical question of whether a configuration learned on an initial training set of data would be effective for new data as it was created (i.e. new requirements and/or new code etc).

RQ4: Are configurations stable over time? An experiment was designed to answer the critical question of how stable the trace configuration is over time. This is an important practical question that addresses the likelihood of thrashing from one configuration to another. This question

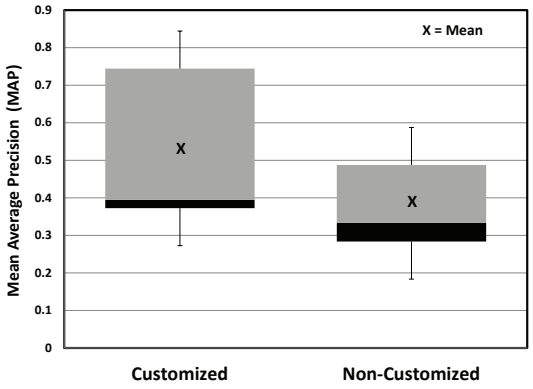


Figure 7: Difference in MAP scores obtained across all evaluated datasets for customized versus non-customized configurations. Maximum, Minimum, Mean, as well as 1st, 2nd (Median), and 3rd Quartiles are shown.

was explored against the Industry-1, Industry-2, and CCHIT datasets, as these were the only datasets of sufficient size to support this type of analysis. We created four versions of the source artifacts including 25%, 50%, 75%, and 100% of the data, labeled D25, D50, D75, and D100 respectively. The sub-parts were derived following the natural structure of the requirements documents that constituted source artifacts in all three datasets. This meant that D25 represented the first physical 25% of the requirements appearing in the specification and D50 represented the first 50% of the requirements, and so on.

The DTC algorithm was used to identify the top configuration for tracing from each subset of source artifacts (i.e. D25, D50, D75, and D100) to the target artifacts shown in Table 1, i.e. in the case of Industry-1, the first trace was executed from the first 25% of requirements to the 3,104 design elements, and so on. The top configurations are shown in Table 4, labeled C25, C50, C75, and C100 respectively.

In the case of the Industry-1 dataset, we see an initial configuration based on *LSI* for the D25 subset and then a transition to *VSM* with use of the global tf-idf computations (i.e. American National Corpus of written terms). The configuration remained relatively stable as the datasize grew over D50-D100. We see a similar pattern for the CCHIT dataset. The initial D25 configuration utilized VSM, but this was replaced by LSI at D50, and from that point on the configuration remained relatively stable. In the final case of Industry-2, the configuration was slightly less stable. We hypothesize that this could have been because the number of source artifacts was significantly smaller than in the other two datasets (i.e. 224 requirements vs. 442 and 453 in Industry-1 and CCHIT respectively), and so the configuration did not have time to stabilize.

These results suggest that changing configurations too frequently during initial phases of the project could lead to thrashing; however some reconfiguration is necessary as the size of the project grows. Although not investigated here, it might be helpful to evaluate configurations over small sized increments of growth, and only reconfigure if a configuration is stable over a window of several increments or if it returns a very significant increase in MAP score.

Table 4: Trace configurations for various sized growth increments of three datasets

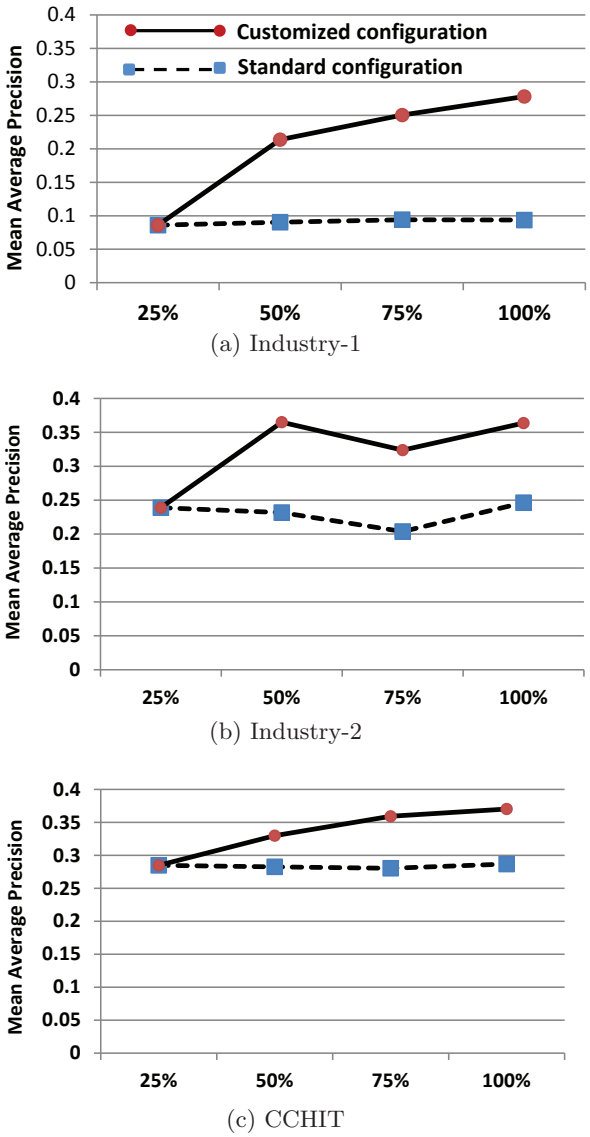
Features	Industry-1				Industry-2				CCHIT			
	C25	C50	C75	C100	C25	C50	C75	C100	C25	C50	C75	C100
STEM	●	●	●	●		●	●	●	●	●	●	●
SourceStandardStop	●						●					
TargetStandardStop	●						●			●	●	●
SourceDynamicStop		●	●	●					●			●
TargetDynamicStop			●	●								
SplitSource						●		●				
SplitTarget					●	●		●				●
SourceAcronym												
TargetAcronym	●					●	●					
Tf-idf Dict Builder	●				●	●	●	●	●	●	●	●
ANC Dict Builder		●	●	●								
Vector Space Model		●	●	●	●	●	●	●	●			
LSI	●									●	●	●
Cosine (not norm)					●	●	●	●				
Cosine (normalized)												
Dice												
Jaccard												
SimpleMatching		●	●	●					●			
Incremental display						●						
Ranked display	●	●	●	●	●		●	●	●	●	●	●

RQ5: Do reconfigurations of the trace infrastructure lead to better trace quality in future traces? While previous experimental results clearly demonstrated that trace configurations can be customized according to the current dataset, it is important to know whether runtime configuration improves the overall quality of trace links in the project. This question directly addresses one of the underlying hypotheses of our work, that performing a "what-if" analysis on past data, serves as an effective predictor of the best configuration as the project data continues to grow in size. In this scenario, a baseline trace configuration is used to generate an initial set of trace links, and then dynamically learned configurations are used throughout the remainder of the project.

For experimental purposes we selected an initial default configuration (C0) that included a stemmer, stopper, Vector Space Model with non-normalized cosine similarity, and ranked ordering of results. While we could have selected a different default configuration, this one was chosen because the selected combination of components is fairly standard across the literature [23] and also performed well in our previous experiments. The documents (D25-D100), and configurations (C25-C100) were reused from Experiment 4.

Configuration $C0$ was used to generate trace links and then to compute MAP scores for $D25$, $C25$ for $D50$, $C50$ for $D75$, and finally $C75$ for $D100$. In this way the system was reconfigured after the simulated arrival of each block of 25% of requirements, and the new configuration was used for tracing purposes until the next configuration occured.

Figure 8 depicts two scenarios for each of the three datasets. In the first scenario the $C0$ baseline configuration was used for the entire dataset, while in the second scenario the trace infrastructure was reconfigured upon arrival of each subsequent 25% block of requirements. The three graphs show that reconfiguration led to marked improvements in MAP scores. The most important observation from this experiment is that in all cases the customized configuration outperformed the static configuration. The 'dip' in MAP scores for Industry-2 is attributed to the fact that the Industry-2 dataset is smaller than the other two data-sets (i.e. had a



(a) Industry-1

(b) Industry-2

(c) CCHIT

Figure 8: Results of Experiment 5 showing the impact of utilizing a default industry standard configuration (C0) versus dynamically reconfiguring the trace infrastructure at regular intervals (i.e. arrival of next 25% of requirements) across the project

total of 224 requirements only) and, therefore, the results were more sensitive to nuances in individual requirements.

5. THREATS TO VALIDITY

There are a number of potential threats to validity. One threat to internal validity is that the top configurations identified in our experiments are based on the selected architectural design, the feature model used in the experiments, and also the components which we implemented. Different choices in any of these areas might have produced different results. The issue of component quality was partially mitigated through reusing as many existing TraceLab [15] components as possible, especially those which had been used successfully in previously published results. For example, the LSI feature was implemented through using an LSI component previously developed by researchers at the College of William and Mary. More importantly, the primary contribution of this paper is the process for dynamically discovering an optimal configuration given an available set of features, and is not primarily designed to comparatively evaluate different algorithms.

A threat to external validity is concerned with the data sets used in the experiments. We assumed that the initial set of trace links provided for the data sets are correct, independent of the way they have been generated. This issue is mitigated in two ways. First, trace links for Industry-1 and Industry-2 were provided by our industry collaborators. The provided links are therefore likely to be correct, although there is no guarantee of completeness. I-Trust links were likewise created by a member of the I-Trust project. Links for the other three datasets have been created by various research groups and have been used extensively in past research projects. Despite potential inaccuracies in the answer set, our approach is designed to search for the best trace configuration in terms of delivered MAP scores, given the known set of trace links and would only be significantly impacted if the quality of the trace links were very low. This is not the case, as each trace matrix has undergone significant review either in an industrial or research setting.

In terms of experimental design, it was not possible to explore every single configuration to know whether the GA algorithm discovered the optimal configuration. Given the running time needed to test certain configurations, we were unable to repeat execution of the entire DCT algorithm multiple times for each experiment in order to perform a more rigorous statistical analysis. We leave this for future work. Furthermore, in the experiments in which we compared the efficacy of different configurations, it is possible, although somewhat unlikely given the consistency of our results, that another undiscovered configuration would have performed well across all datasets.

Finally, only six different projects were used in this work, providing a limited perspective on the traceability problem. This limitation, which is partially due to the significant challenge of acquiring industrial sized datasets, means that we cannot make broad generalizations about the relationships between data characteristics and trace configurations. On the other hand, several of the datasets represent large industrial projects taken from different domains, artifact types, and sizes, and therefore provide a solid context for the experiments.

6. RELATED WORK

In related work, Gethers et al. [19] analyzed the benefits of combining IR techniques such as VSM and the Jensen and Shannon model with Relational Topic Modeling to improve traceability recovery accuracy. Similarly, Chen and Grundy [6] combined VSM with Regular Expressions, Key Phrases, and Clustering techniques. Dekhtyar et al. [13] used voting committees composed of three to five trace retrieval techniques to identify mistakes introduced by human analysts when building a trace matrix. However all of these approaches integrated a fixed set of previously selected techniques. Finally, Falessi et al. [18] empirically evaluated whether a limited combination of techniques outperformed a single technique. They used logistic regression to compute the optimal combination of techniques; unlike DTC, their

approach was not designed to support dynamic configuration of a large and potentially growing set of techniques.

Dynamically adaptable and configurable systems have been the focus of study in several areas of computing such as software engineering, robotics, control systems, programming languages, and bioinformatics. The software engineering community attaches great importance to this topic and has proposed two roadmaps on software engineering for self-adaptive systems [5, 11].

In Requirements engineering, self-adaptive systems include languages to address uncertainty when specifying the behavior of adaptive systems [39], requirements monitoring frameworks to verify violations of system's properties [31], and changes in requirements due to other requirements [34]. Architecture models have also been used to support adaptation of software systems [7, 10, 24]. For example, Dashofy et al. [10] present a framework for self-healing of event-based software architectures which uses "what-if" analysis to verify the change impact in architecture descriptions.

Genetic algorithms (GAs) have been used in different software engineering activities [22]. Examples of these activities include, but are not limited to, evaluation of software reliability [25], refactoring [29], software testing [17, 26, 30], identification of valid design pattern transformation for software reusability [1], software clone detection [37], and configuration of topic modelling techniques [28]. Our approach has similarities to Ensan et al.'s approach [17], which uses a GA to explore the configuration space of product line feature models in order to identify test suites at low size complexity while balancing error and feature coverage. In contrast, our approach is designed to identify an optimal configuration of the traceability infrastructure. While both Ensan's approach and our approach use a GA to search the configuration space of a feature model, the fitness functions are very different. GAs were also used by Wang et al. [37] to identify suitable configurations for clone detection techniques in order to ameliorate the confounding configuration choice problem that is found in several clone detection techniques. The LDA-GA approach [28] uses a GA to configure the parameters of LDA (Latent Dirichlet Allocation) topic modelling technique, in order to improve the performance of traceability link recovery, feature location, and software artefacts labelling. In contrast, our work uses a GA to explore configurations and compositions of different tracing techniques of which LDA could be one of the possible features.

7. CONCLUSION

This paper has presented a novel approach for composing and configuring a trace infrastructure according to the data characteristics of a project. From an implementation perspective, DTC adds little overhead to the human effort involved in the tracing process as all of the functionality can be bundled up and deployed into a single tracing component. While DTC's benefits can only be realized once an initial training set of confirmed trace links has been established, it is primarily during the later phases of development and maintenance that tracing support is most needed. Once realized, the benefits of DTC can lead to significant improvements in the accuracy of the generated trace links.

DTC also has the potential to significantly impact future research practices in the area of traceability. Currently, new techniques are typically compared against a single baseline technique such as VSM or LSI. The authors of this paper have also followed this practice in the past. However, a more rigorous analysis would either compare a new technique against the top configurations for a set of datasets, or would demonstrably show that the technique generally improves trace results across multiple datasets in comparison to a baseline technique such as VSM or LSI.

Finally, DTC supports technology transfer. If an organization has adopted DTC, then there is little innate risk in adding a new feature to the mix. If the feature is effective it will be selected during the configuration process, and if it is not-effective it will not be used. This approach follows a low-risk in-situ approach to data-driven adoption.

This paper has described the DTC approach and conducted an initial series of experiments. However, in future work we intend to extend the feature model to incorporate additional features, and to explore different searching techniques such as hill climbing.

8. ACKNOWLEDGMENTS

The work in this paper was partially funded by US National Science Foundation Grants CNS-0959924 and CCF-0810924. We thank Arnaud Hubaux and Patrick Heymans for providing the TVL tool and Bogdon Dit and Denys Poshyvanyk for sharing their LSI code with us.

9. REFERENCES

[1] M. Amoui, S. Mirarab, S. Ansari, and C. Lucas. A genetic algorithm approach to design evolution using design patterns transformation. *Intel. Comp.*,, 1(2):235–244, 2006.

[2] 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, 2002.

[3] H. U. Asuncion, A. Asuncion, and R. N. Taylor. Software traceability with topic modeling. In *International Conference on Software Engineering*, pages 95–104, 2010.

[4] 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*, 2010.

[5] Betty H. C. Cheng et al. Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems*, pages 1–26, 2009.

[6] X. Chen and J. C. Grundy. Improving automated documentation to code traceability by combining retrieval techniques. In P. Alexander, C. S. Pasareanu, and J. G. Hosking, editors, *ASE*, pages 223–232. IEEE, 2011.

[7] S.-W. Cheng, V. Poladian, D. Garlan, and B. R. Schmerl. Improving architecture-based self-adaptation through resource prediction. In *Software Engineering for Self-Adaptive Systems*, pages 71–88, 2009.

[8] A. Classen, Q. Boucher, and P. Heymans. A text-based approach to feature modelling: Syntax and semantics of tvl. *Sci. Comput. Program.*, 76(12):1130–1143, Dec. 2011.

[9] A. Czauderna, M. Gibiec, G. Leach, Y. Li, Y. Shin, E. Keenan, and J. Cleland-Huang. Traceability challenge 2011: Using tracelab to evaluate the impact

of local versus global idf on trace retrieval. *International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, 6, 2011.

[10] E. M. Dashofy, A. van der Hoek, and R. N. Taylor. Towards architecture-based self-healing systems. In *WOSS*, pages 21–26, 2002.

[11] R. de Lemos, H. Giese, H. Muller, and M. Shaw. Software engineering for self-adaptive systems: A second research roadmap. In *Dagstuhl Seminar proceedings 10431, Software Engineering for Self-Adaptive Systems*, 2010.

[12] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. Enhancing an artefact management system with traceability recovery features. In *ICSM '04: Proceedings of the 20th IEEE International Conference on Software Maintenance*, pages 306–315, Washington, DC, USA, 2004. IEEE Computer Society.

[13] A. Dekhtyar, J. H. Hayes, S. K. Sundaram, E. A. Holbrook, and O. Dekhtyar. Technique integration for requirements assessment. In *RE*, pages 141–150, 2007.

[14] C. Duan and J. Cleland-Huang. Clustering support for automated tracing. In *ASE*, pages 244–253, 2007.

[15] E. Keenan et al. Tracelab: An experimental workbench for equipping researchers to innovate, synthesize, and comparatively evaluate traceability solutions. In *Tool Demo, 34th International Conf. on Software Engineering (ICSE)*, pages 1375–1378, 2012.

[16] M. Eaddy, A. V. Aho, G. Antoniol, and Y.-G. Guéhéneuc. Cerberus: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis. In *ICPC*, pages 53–62, 2008.

[17] F. Ensan, E. Bagheri, and D. Gasevic. Evolutionary search-based test generation for software product line feature models. In *CAiSE*, pages 613–628, 2012.

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

[19] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. DeLucia. On integrating orthogonal information retrieval methods to improve traceability link recovery. In *Proc. of 27th IEEE International Conference on Software Maintenance (ICSM'11)*, pages 133–142, 2011.

[20] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[21] S. Guckenheimer. *Software Engineering with Microsoft Visual Studio Team System*. Adison Wesley, Boston, MA, 2006.

[22] M. Harman, S. A. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.*, 45(1):11, 2012.

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

[24] J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In *FOSE*, pages 259–268, 2007.

[25] A. Kuri-Morales. The application of genetic algorithms to the evaluation of software reliability.

[26] Z. Li, M. Harman, and R. M. Hierons. Search algorithms for regression test case prioritization. *IEEE Trans. Software Eng.*, 33(4):225–237, 2007.

[27] P. Maeder, P. Jones, Y. Zhang, and J. Cleland-Huang. Strategies for effective traceability in safety critical systems. *IEEE Software*, To appear (May/June 2013).

[28] A. Panichella, B. Dit, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *ICSE*, pages 522–531, 2013.

[29] M. D. Penta, M. Neteler, G. Antoniol, and E. Merlo. A language-independent software renovation framework. *Journal of Systems and Software*, 77(3):225–240, 2005.

[30] A. Rathore, A. Bohara, R. G. Prashil, T. S. L. Prashanth, and P. R. Srivastava. Application of genetic algorithm and tabu search in software testing. In *Bangalore Compute Conf.*, page 23, 2011.

[31] W. N. Robinson. A requirements monitoring framework for enterprise systems. *Requir. Eng.*, 11(1):17–41, 2006.

[32] Y. Shin and J. Cleland-Huang. A comparative evaluation of two user feedback techniques for requirements trace retrieval. In *Proc. of 27th Symposium on Applied Computing (SAC)*, 2012.

[33] Y. Shin, J. Huffman Hayes, and J. Cleland-Huang. A framework for evaluating traceability benchmark metrics. In *Technical report, DePaul University, School of Computing*, pages TR:12–001, 2012.

[34] V. Souza, A. Lapouchnian, and J. Mylopoulos. Evolution requirements for adaptive systems. *7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2012.

[35] 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.

[36] S. K. Sundaram, J. H. Hayes, A. Dekhtyar, and E. A. Holbrook. Assessing traceability of software engineering artifacts. *Requir. Eng.*, 15:313–335, September 2010.

[37] T. Wang, M. Harman, Y. Jia, and J. Krinke. Searching for better configurations: A rigorous approach to clone evaluation. In 9^{th} *joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2013)*, 2013. To appear.

[38] E. Weyuker, R. Bell, and T.J.Ostrand. We're finding most of the bugs, but what are we missing? In *3rd International Workshop on Testing, Verification and Validation*, 2010.

[39] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J.-M. Bruel. Relax: a language to address uncertainty in self-adaptive systems requirement. *Requir. Eng.*, 15(2):177–196, 2010.