
Automated Training-Set Creation for Software Architecture Traceability Problem

Waleed Zogaan · Ibrahim Mujhid · Joanna C. S. Santos · Danielle Gonzalez · Mehdi Mirakhorli

Abstract Automated trace retrieval methods based on machine-learning algorithms can significantly reduce the cost and effort needed to create and maintain traceability links between requirements, architecture and source code. However, there is always an upfront cost to train such algorithms to detect relevant architectural information for each quality attribute in the code. In practice, training supervised or semi-supervised algorithms requires the expert to collect several code snippets of architectural tactics that implement a quality requirement and train a learning method. Establishing such training set can take weeks to months to complete. Furthermore, the effectiveness of this approach is largely dependent upon the knowledge of the expert. In this paper, we present three baseline approaches for the creation of training data. These approaches are (i) Manual Expert-Based, (ii) Automated Web-Mining, which generates training sets by automatically mining tactic’s APIs from technical programming websites, and lastly (iii) Automated Big-Data Analysis, which mines ultra-large scale code repositories to generate training sets. We compare the trace-link creation accuracy achieved using each of these three baseline approaches and discuss the costs and benefits associated with them. Additionally, in a separate study, we investigate the impact of training set size on the accuracy of recovering trace links. The results indicate that automated techniques can create a reliable training set for the problem of tracing architectural tactics. Our analysis also indicate that these automated techniques can be used in the other areas of software traceability.

W. Zogaan
Software Engineering Department
Rochester Institute of Technology
Rochester, NY, USA
E-mail: waz7355@rit.edu

I. Mujhid
E-mail: ijm9654@rit.edu

J. C. S. Santos
E-mail: jds5109@rit.edu

D. Gonzalez
E-mail: dng2551@rit.edu

M. Mirakhorli
E-mail: mehdi@se.rit.edu

Keywords Architecture Traceability · Dataset Generation · Architecturally Significant Requirements · Automation

1 Introduction

Software architecture design is the first and the fundamental step towards addressing non-functional requirements such as security, privacy, safety, reliability, and performance [8]. To satisfy such requirements an architect must consider alternate design solutions, evaluate their trade-offs, identify the risks and select the best solution [8]. Such design decisions are often based on well-known architectural tactics [7], defined as reusable techniques for achieving specific quality concerns. Tactics come in many different shapes and sizes [7,8]. For example, reliability tactics provide solutions for fault mitigation, detection, and recovery; performance tactics provide solutions for resource contention in order to optimize response time and throughput, and security tactics provide solutions for authorization, authentication, non-repudiation and other such factors.

Establishing round-trip traceability between quality attributes, architectural tactics, design rationales, and relevant areas of the code, support several software engineering activities. Some of these activities include architecture level change impact analysis, design reasoning, requirements validation, safety-case construction, and long-term system maintenance [26, 30]. For instance, practice has shown that erosion of architecture and architectural qualities often occurs when developers make changes to the code without fully understanding the underlying architectural decisions and their associated quality concerns [27, 30]. However if trace links are available, they can be used to keep developers informed of underlying architectural decisions in order to reduce the likelihood of undermining previous design decisions [28]. It is particularly important to trace architectural decisions in safety-critical systems as these decisions often contribute towards mitigating potential risks and ensuring that the system will operate safely and will meet reliability, availability and dependability requirements. [12].

In prior work we proposed Archie [25,28,30], an automated technique based on a data mining approach to trace quality requirements through architectural tactics to source code. Archie included a set of code-based classifiers constructed to detect different architectural tactics in the source code [25,30]. Archie's individual classifiers have been trained to detect audit, asynchronous method invocation, authentication, checkpoint, heartbeat, role-based access control (RBAC), resource pooling, scheduler, and secure session tactics. The classifiers were trained using code snippets of different architectural tactics collected from hundreds of high-performance, open source projects.

While the results were promising [25, 30], we observed that extending this approach to a larger number of tactics requires significant involvement of experts to create new training-sets. This prevents the technique from being practically applied in an industrial traceability setting. This is a significant barrier for software architecture traceability and software traceability in general.

Collecting tactical code snippets from real systems requires a deep understanding of quality attributes and architectural tactics, as well as how these tactics can be implemented. This can be challenging if students or less experienced developers are involved in establishing the training set. Furthermore, even when experts (e.g. architects) are involved in the process of creating such datasets, *the process is very time consuming* as they need to search across various systems, understand their code snippets and select those which are the best representative of the tactics. As a result of such barriers, the community is conducting research using very small, student-generated datasets or limited industrial datasets which are not shareable, impacting both generalizability and reproducibility of research results.

1.1 Contribution

In this paper we present an empirical study and novel techniques that advances our previous work as well as of future software traceability research in several important ways. We first develop new approaches based on (i) *Web-Mining* and (ii) *Big-Data Analysis* to automate the creation of traceability dataset. Web-Mining technique generates training sets by automatically mining tactic's APIs from technical programming websites. In contrast, Big-Data Analysis technique uses an ultra-large scale code repository established in this work to automatically generate quality training sets. The code repository we have established for this paper, contains over 116,000 open source projects.

As the second contribution, we report a series of empirical studies conducted to compare the accuracy of a traceability technique trained using the automatically generated training-sets versus the datasets which are manually established by the experts.

As the third contribution, we provide an on-line tool called BUDGET (available on-line¹) that implements our automated approaches. BUDGET enables the traceability researchers to mine software repositories of 22 million source files to create training sets. BUDGET also implements several data sampling techniques.

Along with development of novel techniques, we report the results of empirical studies designed to investigate the following research questions:

Research Question 1: Does the Training Method Based on Automated-Web Mining Result in Higher Trace-Links Classification Accuracy Compared to Expert Created Training Set?

Through a set of experiments reported in section 4 we investigated this research question. Our empirical analysis indicates that the web-mining approach presented in this paper, produces high quality training set. The accuracy of trace-link classifier trained using the web-mining approach is comparable to the classifier trained using expert-created dataset. The statistical analysis shows that the differences are not statistically significant. This finding can expand the current state of software architecture traceability, by facilitating the creation of training data through use of our proposed automated technique.

Research Question 2: Does the Training Method Based on Automated Big-Data Analysis Result in Higher Trace-Links Classification Accuracy Compared to Expert Created Training Set?

The results of our empirical study indicate that the proposed novel Big-Data Analysis approach creates high quality training-set. The accuracy of trace-link classifier trained using the Big-Data Analysis approach in three experiments out of five was better than an the classifier trained using expert-created dataset. The differences between the accuracy of these two training methods is not statistically significant. Therefore the Big-Data Analysis approach can be used to help researchers create high quality datasets of architectural code snippets. Manually creating such dataset is very time consuming and our automated technique provides a significant reduction in training set creation time

Research Question 3: What is the Impact of Training Set Size on the Accuracy of Trace Link Classification?

¹ <http://design.se.rit.edu/budget/>

Through this research question we aim to perform a cost-benefit analysis for the cases where the training-set is established manually by experts. The goal is to investigate whether it is worth the effort to manually create large training set with the hope of achieving higher accuracy. In an experiment we compared the trace link classification accuracy of a classifier trained using 10, 20, 30, 40 and 50 projects. The results indicate that there is not a significant difference in the accuracy of the classifier for different training set sizes.

Research Question 4: Can Automated Training-Set Creation Approaches Be Applied to the Other Traceability Scenarios?

While we used our automated dataset generation techniques for creating training set in the software architecture traceability domain, it can be applied in other domains as well. To investigate that, we ran a feasibility study which, while not the main contribution of this paper, highlights the directions for future work in similar research area.

1.2 Originality and Extension

This work differs from our previous ICSE 2012 and TSE 2015 publications in different ways. In those papers [24, 30] we proposed the original classification technique used to trace architectural tactics in the source code. In this paper we recognize training set creation as a problem in supervised software traceability approaches including our previous architecture traceability technique. Then we present novel Web-Mining and Big-Data analysis techniques which can help software traceability researchers in establishing training sets. In a series of experiment we evaluate the accuracy of automated dataset generation techniques.

1.3 Organization of the Paper

The remainder of this paper is organized as following: Section 2 provides the background for the architecture traceability problem and the related work in the area of automated dataset creation. In section 3 and 4 we present the overview of the three training set creation techniques and then present the results of our comparison study. We also study the impact of training-set size on accuracy of creating trace links in section 6. Section 7 provides additional usage scenarios where the proposed automated techniques can be used. Section 8 briefly present our tool BUDGET. Section 9 discusses the steps toward reproducibility of the results, while section 10 discusses threats to validity. Finally, we present conclusions and future work in Section 11.

2 Background

This section provides the background for the problem of architecture traceability. It summarizes the related work in the area of automated dataset creation and dataset augmentation.

2.1 Architectural Tactics

Tactics serve as a building block of software architecture and are used to satisfy a specific quality. A formal definition of tactics is provided by Bachman et al. [7] who define a tactic as a “means of satisfying a quality-attribute-response measure by manipulating some aspects of a quality attribute model through architectural design decisions”.

We limited the focus of this work to five tactics: *heartbeat*, *scheduling*, *resource pooling*, *authentication*, and *audit trail*. These were selected because they represent a variety of reliability, performance, and security requirements. They are defined as follows [8]:

- **Heartbeat:** A reliability tactic for fault detection, in which one component (sender) emits a periodic heartbeat message while another component listens for the message (receiver). The original component is assumed to have failed when the sender stops sending heartbeat messages. In this situation, a fault correction component is notified.
- **Scheduling:** Resource contentions are managed through scheduling policies such as FIFO (First in first out), fixed-priority, and dynamic priority scheduling.
- **Resource pooling:** Limited resources are shared between clients that do not need exclusive and continual access to a resource. Pooling is typically used for sharing threads, database connections, sockets, and other such resources. This tactic is used to achieve performance goals.
- **Authentication:** Ensures that a user or a remote system is who it claims to be. Authentication is often achieved through passwords, digital certificates, or biometric scans.
- **Audit trail:** A copy of each transaction and associated identifying information is maintained. This audit information can be used to recreate the actions of an attacker, and to support functions such as system recovery and nonrepudiation.

In the following subsection we present a classification-based approach for tracing architectural tactics in the source code. Through the rest of the paper, we train this classifier using datasets generated by different training set creation techniques and compare the accuracy of generated trace links.

2.2 Traceability Challenge: Identifying Tactic-Related Classes

In previous work [25, 30] we presented a novel approach for tracing architecturally significant concerns, specifically those which are implemented through the use of common architectural tactics. As a tactic is not dependent upon a specific structural format, we cannot use structural analysis as the primary means of identification. Our approach therefore relied primarily on information-retrieval (IR) and machine learning techniques to train a classifier to recognize specific terms that occur commonly across implemented tactics. The tactic-classifier was used to identify all classes related to a given tactic, and then establishes tactic-level traceability to the driving quality requirements and design rationales [27]. The classifier includes three phases of preparation, training, and classification which are defined as follows:

Data Preparation Phase: All data is preprocessed using standard information retrieval techniques (stemming, stop terms removal, etc), and are represented as a vector of terms.

Training Phase: The training phase takes a set of preclassified code segments as input, and produces a set of indicator terms that are considered to be representative of each tactic type. For example, a term such as *priority* is found more commonly in code related to the *scheduling* tactic than in other kinds of code, and therefore receives a higher weighting with respect to that tactic.

More formally, let q be a specific tactic such as *heart beat*. Indicator terms of type q are mined by considering the set S_q of all classes that are related to tactic q . The cardinality of S_q is defined as N_q . Each term t is assigned a weight score $Pr_q(t)$ that corresponds to the probability that a particular term t identifies a class associated with tactic q . The frequency $freq(c_q, t)$ of term t in a class description c related with tactic q , is computed for each tactic description in S_q . $Pr_q(t)$ is then computed as:

$$Pr_q(t) = \frac{1}{N_q} \sum_{c_q \in S_q} \frac{freq(c_q, t)}{|c_q|} * \frac{N_q(t)}{N(t)} * \frac{NP_q(t)}{NP_q} \quad (1)$$

Classification Phase: During the classification phase, the indicator terms computed in Equation 1 are used to evaluate the likelihood ($Pr_q(c)$) that a given class c is associated with the tactic q . Let I_q be the set of indicator terms for tactic q identified during the training phase. The classification score that class c is associated with tactic q is then defined as follows:

$$Pr_q(c) = \frac{\sum_{t \in c \cap I_q} Pr_q(t)}{\sum_{t \in I_q} Pr_q(t)} \quad (2)$$

where the numerator is computed as the sum of the term weights of all type q indicator terms that are contained in c , and the denominator is the sum of the term weights for all type q indicator terms. The probabilistic classifier for a given type q will assign a higher score $Pr_q(c)$ to class c that contains several strong indicator terms for q .

2.3 Related Work on Dataset Creation/Augmentation

There have been many works in the area of data mining and information retrieval to facilitate training set *selection* in text classification problems. However, the fundamental assumptions in this line of research is that a large number of labelled data points exists and these approaches try to incorporate various sampling [33, 35], instance selection [10, 11, 17] and data reduction techniques [31, 35] to obtain a small representative sample. Unlike these approaches, we do not make such assumption and the main problem in the area of software traceability is the lack of any labeled data.

Web-mining has been previously used in the area of requirements traceability, for example, Jane Cleland-Huang et.al. [19] have used web-mining to replace a hard to retrieve query with an augmented query obtained from the web. This work does not focus on creating dataset; it is about expanding the terms in a requirements with more domain specific terms, so it can be better detected by traceability techniques. In contrast, our approach is designed to create code based training set for architectural tactics. Similarly Anas Mahmoud [22] have used query augmentation techniques based on text clustering to classify non-functional requirements. In this approach the terms in the specification of non-functional requirements are augmented with the semantically similar terms extracted from the content of Wikipedia.

Recently there have been a number of projects in the area of mining ultra-large-scale open source software repositories [16, 36], these works primarily focus on studying source code and coding issues. There is a limited experimental research on using such resource to generate scientific datasets, particularly for requirements and architecture research. To the best of our knowledge there is no concrete automated technique to help scientist generate their datasets.

Several independent software engineering communities are providing mechanisms for publishing and sharing datasets. Mining Software Repositories (MSR) conference holds a Data Track every year where researchers can publish and share their dataset, Center of Excellence for Software Traceability holds

traceability challenges where researchers can share their datasets related to a software traceability challenge. Most works published on these repositories are based on manually created datasets [21]. In our work, we utilize massive amount of public data on the web and large scale software repositories and provide required automation to create high quality datasets.

3 Overview of the Three Baseline Techniques

As previously stated, this paper presents novel automated techniques to create training sets for the problem of tracing architectural tactics. These automated techniques are designed to create software traceability datasets with little or no upfront cost while achieving similar (or better) quality than datasets established by experts.

The proposed automated training set creation techniques as well as the traditional expert-based approach are illustrated in Figure 1. In case of Manual Expert-Based approach, architects collect, review and refine the training set. In the case of automated techniques, a description of the tactic from textbooks (or a set of tactic related terms) can be used as a search query. Then, in each approach, advanced searching and filtering techniques are used to identify API descriptions or actual implementation of the tactic from technical libraries or open source software repositories.

In the following subsections we describe each of these baseline techniques. Then in section 4 we report empirical studies conducted to compare them.

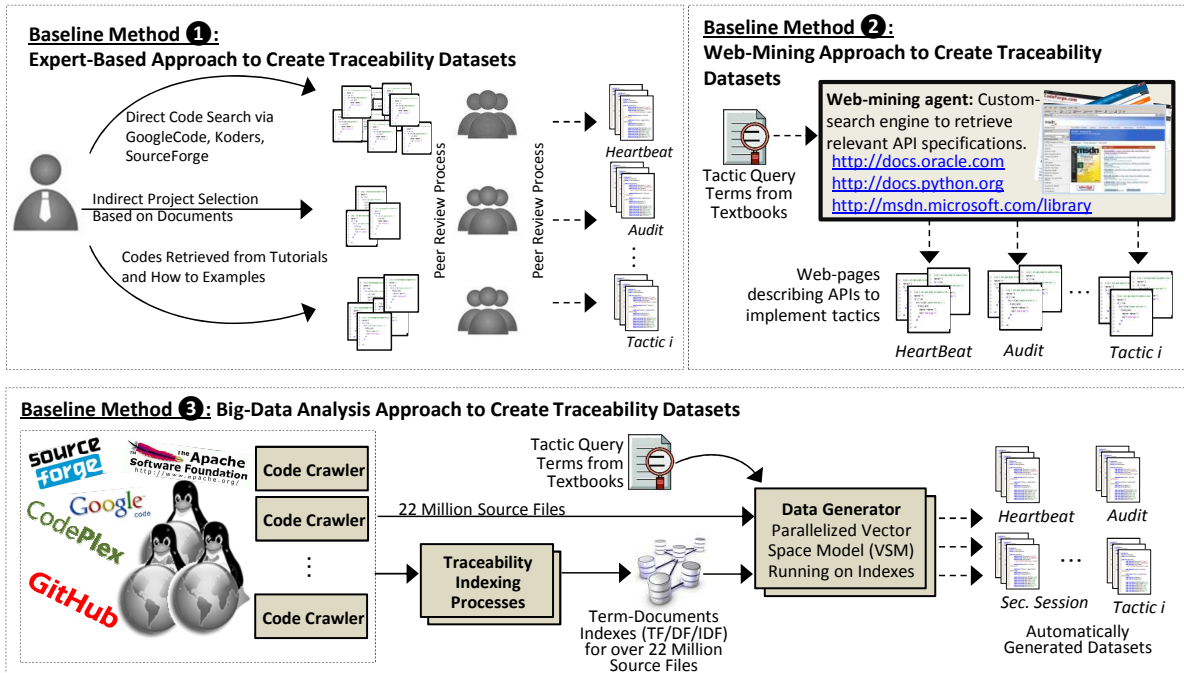


Fig. 1: Overview of Automated Approaches to Create Tactic Traceability Training-sets

3.1 Baseline Method 1: Expert-Centric Approach

In previous work [29, 30] we used a manual approach to collect datasets to train our tactic classifier. The training set shown in Table 2 was established by experts in the area of software architecture and requirements engineering. Then this dataset was peer reviewed and evaluated by two additional independent evaluators. The subject matter experts involved in the project had two to eight years of experience as software architects. The dataset of code snippets implementing architectural tactics were discovered through the following process:

- *Direct Code Search* : The source code search engine *Koders* [3] was used to search for the tactic. The search query for each tactic was composed from keywords used in descriptions of the tactic found in textbooks, articles, and white papers or the libraries that architects have previously used to implement the tactics. All the returned code snippets were reviewed by two other experts to determine whether they were relevant (i.e. related to the current architectural tactic) or not.
- *Indirect Code Search* : Project-related documents, such as design documents, online forums, etc. were searched for references and pointers to architectural tactics. This information was then used to identify and retrieve relevant code. Similarly all the retrieved code snippets were peer-reviewed to ensure that they were implementing the targeted tactic.
- *"How to" examples* : Online materials, libraries (e.g. MSDN), technical forums (such as Stack Overflow) and tutorials were used to extract concrete examples of implemented architectural tactics.

The rigorous search and validation approach used in this manual data collection resulted in a high quality and precise traceability dataset. However, the cost associated with this approach is substantially high. For instance, it took us about 3 months to collect and peer review tactical data from 10 different projects for 5 architectural tactics.

3.2 Web-Mining Approach

Web based libraries, such as *msdn*² or *oracle*³, are one of the resources which contain a rich set of information about implementing architectural tactics as well as many other design and programming concerns. Our initial hypothesis was that creating training sets from these libraries will result in a high quality training set for the classifiers. Figures 2(a) and 2(b) illustrates sample implementation guidelines retrieved from these libraries to implement reliability requirements through *Heartbeat* and security requirements through *Audit Trail* tactics.

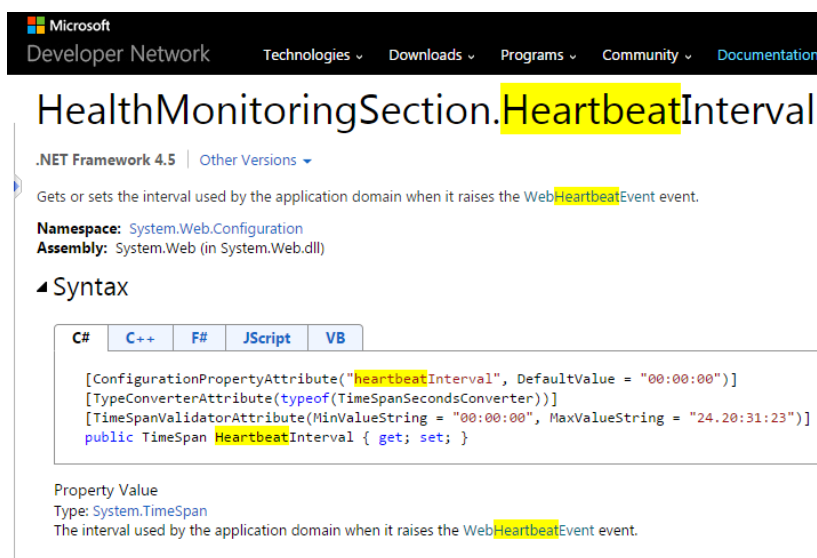
3.2.1 Data Collection Agent

We developed a custom web scraper which uses the search engine APIs of Google to query the content of predefined technical libraries (e.g. *msdn* and *oracle*).

The search query used in this approach contains keywords describing the tactic (drawn from descriptions of the tactic found in textbooks). For example to find APIs related to *HeartBeat* tactic, we used the following textual description from a book [8]: “*Heartbeat is a **fault detection** mechanism that employs a periodic message exchange between a system **monitor** and a process being monitored.*” The

² <https://msdn.microsoft.com>

³ <http://www.oracle.com>



Microsoft
Developer Network Technologies ▾ Downloads ▾ Programs ▾ Community ▾ Documentation

HealthMonitoringSection.HeartbeatInterval

.NET Framework 4.5 | Other Versions ▾

Gets or sets the interval used by the application domain when it raises the WebHeartbeatEvent event.

Namespace: System.Web.Configuration
Assembly: System.Web (in System.Web.dll)

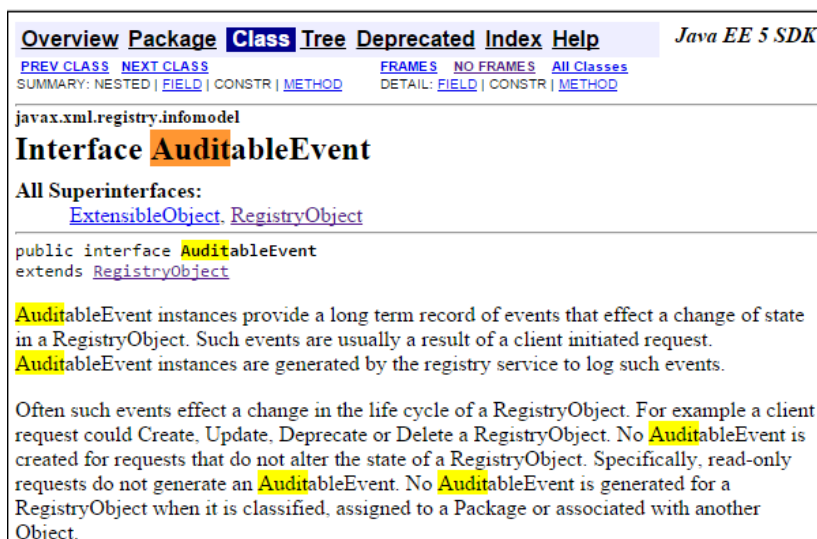
▲ Syntax

C# C++ F# JScript VB

```
[ConfigurationPropertyAttribute("heartbeatInterval", DefaultValue = "00:00:00")]
[TypeConverterAttribute(typeof(TimeSpanSecondsConverter))]
[TimeSpanValidatorAttribute(MinValueString = "00:00:00", MaxValueString = "24.20:31:23")]
public TimeSpan HeartbeatInterval { get; set; }
```

Property Value
Type: System.TimeSpan
The interval used by the application domain when it raises the WebHeartbeatEvent event.

(a) implementing reliability concerns through Heartbeat tactic from msdn.com



Overview Package **Class** Tree Deprecated Index Help Java EE 5 SDK

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [All Classes](#)
SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

javax.xml.registry.infomodel

Interface AuditableEvent

All Superinterfaces:
[ExtensibleObject](#), [RegistryObject](#)

public interface **AuditableEvent**
extends [RegistryObject](#)

AuditableEvent instances provide a long term record of events that effect a change of state in a RegistryObject. Such events are usually a result of a client initiated request. **AuditableEvent** instances are generated by the registry service to log such events.

Often such events effect a change in the life cycle of a RegistryObject. For example a client request could Create, Update, Deprecate or Delete a RegistryObject. No **AuditableEvent** is created for requests that do not alter the state of a RegistryObject. Specifically, read-only requests do not generate an **AuditableEvent**. No **AuditableEvent** is generated for a RegistryObject when it is classified, assigned to a Package or associated with another Object.

(b) addressing security concerns through Audit Trail tactic from Oracle.com

Fig. 2: Two sample API descriptions from technical libraries of (a) MSDN and (b) Oracle

trace user generated the following *trace query* from this description: *Heartbeat OR fault OR detection OR monitoring*.

For each tactic, a number of highly-relevant web pages were collected. The scraper-agent returns the ranked web-pages containing relevant API documentations and sample codes to implement the tactic.

The information within each Web page is filtered, so the HTML tags are removed and only textual content is stored in a plain text file.

3.2.2 Generated Data

For the purpose of training a classification technique, the generated data contains balanced sample text files (web page contents) that are either tactic-related (positive samples) or non-tactical (negative samples). Although the Web-Mining approach is able to generate unbalanced training sets, for the sake of comparing different baseline techniques we generate balanced datasets.

The positive samples are API documentations for a tactic or sample tactical code snippets. The negative or non-tactical samples are sets of documents which have the highest dissimilarity to the originated query. Negative samples would help to remove the terms which are dominant in the Web pages of the library (e.g. Microsoft in MSDN library).

3.3 Big-Data Analysis Approach

This approach relies on using machine learning approaches to create the code-based training sets by mining ultra large scale open source repositories. Our approach includes several different components as illustrated in Figure 1.

3.3.1 Creating Ultra-Large Scale Repository of Open Source Projects

The first component is the source code scraper, responsible for mining source code of projects from a wide range of open source repositories.

For the purpose of this study, we have extracted over 116,609 projects from Github, Google Code, SourceForge, Apache, and other software repositories. We have developed different code crawling applications to extract projects from all these different code repositories. To extract the projects from Github, we make use of a torrent system known as GHTorrent⁴ that acts as a service to extract data and events and gives it back to the community in the form of MongoDB data dumps. The dumps are composed of information about projects in the form of users, comments on commits, languages, pull requests, follower-following relations, and others.

We also utilized *Sourcerer* [34], an automated crawling, parsing, and fingerprinting application developed by researchers at the University of California, Irvine. *Sourcerer* has been used to extract projects from publicly available open source repositories such as Apache, Java.net, Google Code and Sourceforge. The *Sourcerer* repository contains versioned source code across multiple releases, documentation (if available), project metadata, and a coarse-grained structural analysis of each project. We have downloaded the entire repository of open source systems from these code repositories.

After having extracted all these projects from Github and other repositories, we performed a data cleaning where we removed all the empty or very small projects. Table 1 shows the frequency of all the projects in different languages in our repository.

⁴ <http://ghtorrent.org/>

Table 1: Overview of the projects in Source Code Repository of Big-Data Analysis Approach

Language	Freq.	Language	Freq.	Language	Freq.	Language	Freq.	Language	Freq.
Java	32191	Go	1614	Emacs Lisp	321	ActionScript	120	F#	74
JavaScript	22321	CoffeeScript	1187	Visual Basic	134	Elixir	82	Kotlin	43
Python	9960	Scala	729	Erlang	154	Scheme	80	Bison	39
CSS	9121	Perl	699	Processing	152	Prolog	77	Cuda	37
Ruby	8723	Arduino	321	PowerShell	151	D	72	LiveScript	32
PHP	8425	Lua	458	TypeScript	139	Common Lisp	65	AGS Script	29
C++	5271	Clojure	456	OCaml	105	Pascal	60	SQLF	26
C	4592	Rust	308	XSLT	102	Haxe	60	Mathematica	25
C#	4230	Puppet	286	ASP	85	FORTRAN	45	Apex	22
Objective-C++	33	Groovy	253	Dart	84	OpenSCAD	44	PureScript	22
Objective-C	2616	SuperCollider	185	Julia	84	Racket	44	DM	21

*Total number of projects:116,609, *Total number of source files: 23M

3.3.2 Indexing the Data

The second component of the Big-Data Analysis approach is a term-document indexing module, which indexes the occurrence of terms across source files of each project in our code repository. This component, which is called *Traceability Indexing*, first pre-processes each source file, removes the stop words, stems the terms to its root form and then indexes source files. The index stores statistics about each documents (source files) such as *term frequency (TF)*, *document frequency (DF)*, *TF/IDF* and *location of source file* in order to make term-based search more efficient. This is an inverted index which can list, for a term, the source files that contain it [23].

3.3.3 Data Generator Component

The third component is a paralleled version of Vector Space Model (VSM) [32] capable running over 22 million source files in a few seconds. VSM is a standard approach which computes the cosine similarity between a query and document, each of which is represented as a vector of weighted terms. A more complete explanation is provided in most introductory information retrieval textbooks [32].

This component is used to generate a tactical dataset based on a query provided by a trace user. It calculates the cosine similarity score between provided query and all the source files in the ultra large scale software repository. For each tactic, the most relevant source files exhibiting highest similarity to the trace query are selected. In order to avoid domain specific files, this component also retrieves n samples of non-tactical files for each tactic from the same project (n is defined by the user). Previously it has been proven that unrelated sample data has significant impact on quality of trained indicator terms for the classifier presented in this paper [15, 25, 30].

3.4 Generated Data

The generated data contains a balanced dataset of tactical and non-tactical code snippets retrieved from 10 open source projects. From each project, a tactical file and one non-tactical file is retrieved.

4 Experiment Design

This section presents the experiment design to compare three baseline training-set creation techniques and to answer our research questions.

In the following we describe the justification for selection of these techniques, and the details of the methodology used to conduct the comparison and validate the results.

4.1 Justification for Selection of Approaches

The domain of automatically-generated training sets is relatively new. Although there are previous studies on trace-query replacement and augmentation, the idea of automatically generating training-set has not been explored.

Development of such approaches relies on the existence of large, (un)structured and rich knowledge bases. Since both Web and ultra-large-scale code repositories have such characteristics, one key novelty of the proposed work in this paper is to utilize such resources and develop new techniques to help scientists in the area of software architecture traceability to obtain high quality datasets.

4.2 Oracle Dataset Used as Testing set

The expert-created dataset of architectural tactics was used as the testing-set and a measurement for comparison of the three baseline techniques. This dataset was manually collected and peer reviewed by experts over the time frame of three months.

For each of the five tactics, the experts have identified 10 open-source projects in which the tactic was implemented. For each project, they performed an architectural biopsy to retrieve a source file in which the targeted tactic was implemented and also retrieved one randomly selected non-tactical file. Using this data we built a balanced training set for each tactic which included 10 tactic-related snippets and 10 non-tactical ones.

4.3 Experiment Design

Three different experiments were designed to answer research questions related to comparison of baseline techniques.

4.3.1 *Experiment Design for Using Baseline Method 1*

The accuracy of classification techniques trained using Expert-Based approach was evaluated using a standard 10-fold cross-validation process. In this experiment the expert created code-snippets dataset served as both the training and testing set. This is a classic evaluation technique widely used in the area of data mining and information retrieval and automated requirements traceability [13, 14, 20, 30].

In each execution, the data was partitioned by project such that in the first run nine projects, each including one related and four unrelated code-snippets, were used as the training set and one project was used for testing purposes. Following ten such executions, each of the projects was classified one time. The experiment was repeated using the same pairs of term thresholds and classification thresholds used in the previous experiment.

Table 2: Manual Dataset Generated by Expert

Tactic	Projects
Audit	1-Jfolder(Programming), 2-Gnats(Bugs Tracking), 3-Java ObjectBase Manager(Database), 4-Enhydra Shark(Business, workflow engine), 5-Openfire aka Wildfire(Instant messaging), 6-Mifos(Financial), 7-Distributed 3D Secure MPI(Security), 8-OpenVA.(Security), 9-CCNetConfig(Programming), 10-OAJ (OpenAccountingJ)(ERP)
Scheduling	1-CAJO library(Programming), 2-JAVA DynEval(Programming), 3-WEKA Remote Engine(Machine Learning), 4-Realttime Transport Protocol(Programming), 5-LinuxKernel(Operating Systems), 6-Apache Hadoop(Parallel Computing), 7-ReactOS(Operating Systems), 8-Java Scheduler Event Engine(Programming), 9-XORP(Internet Protocol), 10-Mobicents(Mobile Programming)
Authentication	1-Alfresco(Content management), 2-JessieA Free Implementation of the JSSE(Security), 3-PGRADE Grid Portal(Business, workflow engine), 4-Esfinge Framework(Programming), 5-Classpath Extensions(Workflows Management), 6-Jwork(Programming), 7-GVC.SiteMaker(Programming), 8-WebMailJava(Programming), 9-Open Knowledge Initiative(OKI)(Education), 10-Aglet Software Development Kit(Programming)
Heartbeat	1- Real Time Messaging-Java(Programming), 2-Chat3(Instant messaging), 3-Amalgam(Content Management), 4-Jmmp(Programming), 5-RMI Connector Server(Web Programming), 6-SmartFrog(Parallel Computing), 7-F2(Financial), 8-Chromium NetworkManager(Web Programming), 9-Robot Walk Control Behavior(Programming), 10-Apache(Programming)
Pooling	1-ThreadPool Class(Programming), 2-Open Web Single Sign On(Web Programming), 3-ThreadStateMapping2(Programming), 4-RIFE(Web Programming), 5-Mobicents(Mobile Programming), 6-Java Thread Pooling Framework(Programming), 7-Concurrent Query(Programming), 8-RIFE(Web Programming), 9-RIFE(Web Programming), 10-EJBs(Web Programming)

4.3.2 Experiment Design for Using Baseline Method 2

In the second baseline approach we used a web-mining technique to automatically extract data from technical libraries such as *MSDN* and *ORACLE*. The tactic classifier was trained using this dataset, and then tested against the accurate dataset of code snippets established by experts (table 2). The experiment was repeated using a variety of term thresholds and classification thresholds required for formulas 1 and 2.

4.3.3 Experiment Design for Using Baseline Method 3

Last baseline method was trained by the training set generating using Big-Data Analysis approach. Then the trained classifier was used against the oracle dataset of tactical code snippets collected by the experts. The training data was sampled from over 116,000 open source projects in our code repository.

4.4 Evaluation Metrics

Results were evaluated using four standard metrics of recall, precision, f-measure, and specificity computed as follows where *code* is short-hand for *code snippets*.

$$Recall = \frac{|RelevantCode \cap RetrievedCode|}{|RetrievedCode|} \quad (3)$$

while precision measures the fraction of retrieved code snippets that are relevant and is computed as:

$$Precision = \frac{|RelevantCode \cap RetrievedCode|}{|RetrievedCode|} \quad (4)$$

Because it is not feasible to achieve identical recall values across all runs of the algorithm the F-Measure computes the harmonic mean of recall and precision and can be used to compare results across experiments:

$$FMeasure = \frac{2 * Precision * Recall}{Precision + Recall} \quad (5)$$

Finally, specificity measures the fraction of unrelated and unclassified code snippets. It is computed as:

$$Specificity = \frac{|NonRelevantCode|}{|TrueNegatives| + |FalsePositives|} \quad (6)$$

4.5 Minimizing Biases

To avoid the impact of dataset size, all the automatically generated datasets included 10 projects, (or 10 related web-pages). We trained the classifier using the code snippets automatically extracted using our own primitive big-data analysis technique and then attempted to classify the accurate dataset of manually established and reviewed code snippets.

In order to avoid the bias of datasets size and primarily comparing the quality of training sets, we decided to use the dataset size equal to manual training-set. Therefore, we only included 10 sample API specifications. Furthermore, for training purposes, similar to manual case, this dataset also includes 40 descriptions of non-tactic-related IT documents collected by our web-scrapers.

To minimize the biases toward selection of terms in the tactic query, we solicited terms from text book descriptions of the tactic. More systematic approaches were conducted to address other related threats to validity, which are thoroughly discussed in section 10.

5 Results

The experiments design described in section 4 was followed to train the tactic classifier using three baseline approaches and compare the results. Table 3 shows the top ten indicator terms that were learned for each of the five tactics using the three training techniques. While there is significant overlap, the code-snippet approaches unsurprisingly learned more code-oriented terms such as *ping*, *isonlin*, and *pwriter*.

Figure 3 reports the f-measure results for classifying classes by tactic using several combinations of threshold value. Overall three baseline methods obtained similar accuracy. In two cases, namely *audit* and *heartbeat* the classifier trained using Expert collected code-snippets outperformed the classifier trained using automated techniques. In case of *authentication* the classifier trained using the manually collected dataset achieved the same level of accuracy as BigData-trained classifier.

In the case of *pooling* and *scheduling*, the Big-data-trained classifier outperformed the other approaches at term threshold values of 0.01 and 0.001 and classification thresholds of 0.7 to 0.3. One phenomenon that needs explaining in these graphs is the horizontal lines in which there is no variation in f-measure score across various classification values. This generally occurs when all the terms scoring over the term threshold value also score over the classification threshold.

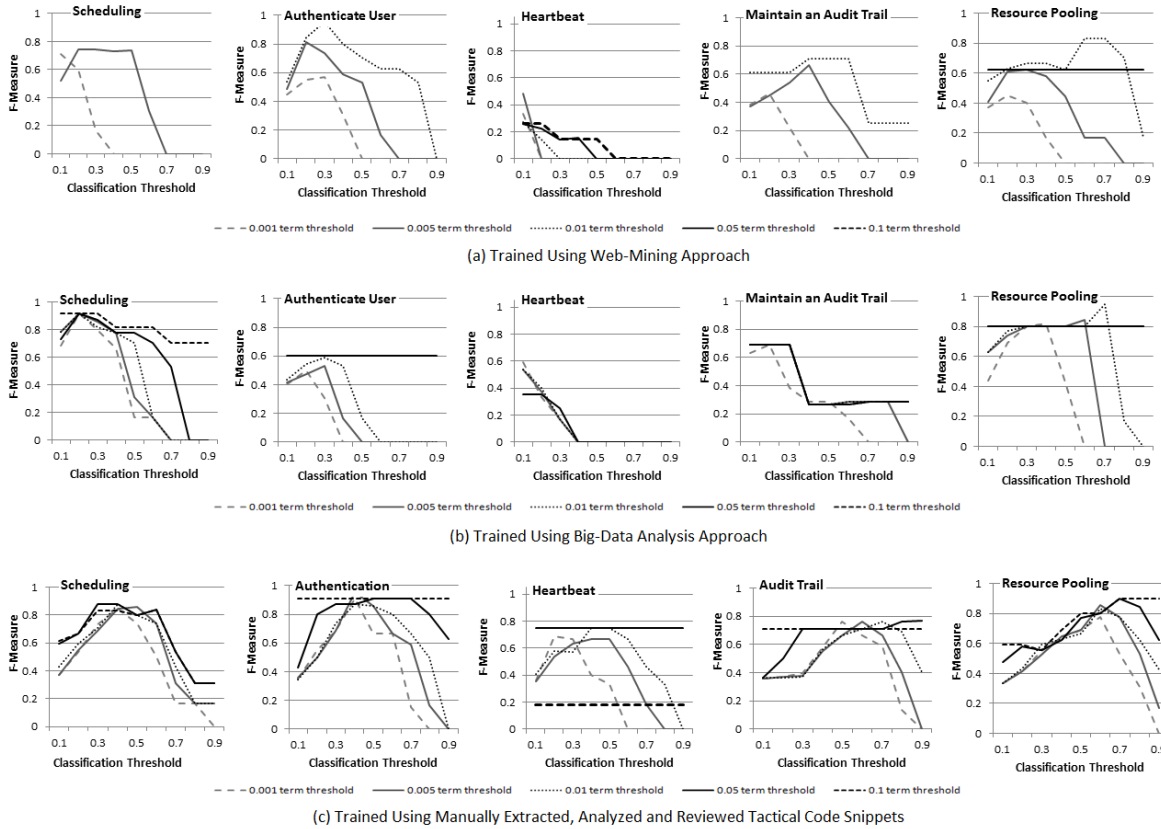


Fig. 3: Results for Detection of Tactic-related Classes at various Classification and Term Thresholds for five Different Tactics

Table 4 reports the optimal results for each of the tactics i.e. a result which achieved the high levels of recall (0.9 or higher if feasible) while also returning as high precision as possible. The results show that in four cases the classifier trained using manually collected data recalled the entire tactic related classes, while also achieving reasonable precision.

The BigData-trained classifier achieved recall of 0.909 in one case and recall of 1 for two of the tactics. The classifier trained using Web-based approach achieved recall of 1, in two cases and 0.909 for two other tactics.

RQ 1: Manual Expert-Based Training vs. Automated Web-Mining.

The above results indicate that, in four out of five cases the manual Expert-Based approach outperformed the Web-Mining technique. However, the differences were very small. Table 5 shows the differences between f-measure of Expert-Based Approach and Web-Mining.

Based on this limited observation, we can rank Expert-Based baseline method equivalent to the Manual approach. In order to evaluate whether differences were statistically significant we performed *Wilcoxon* tests as well as the *Friedman ANOVA* test which is a non-parametric test for comparing

Table 3: Indicator terms learned during training

Tactic Name	Web-Mining trained indicator terms	Big-Data trained indicator terms	Code trained indicator terms
Heartbeat	nlb cluster balanc wlb ip unicast network subnet heartbeat host	counter, fd, hb, heartbeat, member, mbr, suspect, ping, hdr, shun	heartbeat, ping, beat, heart, hb, outbound, puls, hsr, period, isonlin
Scheduling	schedul parallel task queue partition thread unord ppl concurr unobserv	schedul, priorit, task, feasibl, prio, norm, consid, paramet, polici, thread	schedul, task, priorit, prcb, sched, thread, , rtp, weight, tsi
Authentication	authent, password, user, account, credenti, login, membership, access, server, sql	password, login, usernam, rememb, form, authent, persist, sign, panel, succeed	authent, credenti, challeng, kerbero, auth, login, otp, cred, share, sasl
Resource Pooling	thread, wait, pool, applic, perform, server, net, object, memori, worker	pool, job, thread, connect, idl, anonym, async, context, suspend, ms	pool, thread, connect, sparrow, nbp, processor, worker, time-wait, jdbc, ti
Audit Trail	audit, transact, log, sql, server, secur, net, applic, databas, manag	trail, audit, categori, observ, udit, outcom, ix, bject, acso, lesser	audit, trail, wizard, pwriter, lthread, log, string, categori, pstmt, pmr

Table 4: A Summary of the Highest Scoring Results

Tactic	Training Method	FMeasure	Recall	Prec.	Spec.	Term/ Classification threshold
Audit	Web-Mining	0.71	1	0.55	0.785	0.01 / 0.4
	Big-Data	0.687	1	0.523	0.762	0.001 / 0.2
	Expert-Manual	0.758	1	0.611	0.833	0.001 / 0.5
Authentication	Web-Mining	0.956	1	0.916	0.9772	0.01 / 0.3
	Big-Data	0.6	0.545	0.666	0.931	0.05 / 0.1
	Expert-Manual	0.956	1	0.916	0.977	0.005 / 0.4
Heartbeat	Web-Mining	0.48	0.545	0.428	0.813	0.005 / 0.1
	Big-Data	0.592	0.727	0.5	0.813	0.001 / 0.1
	Expert-Manual	0.689	1	0.526	0.775	0.001 / 0.2
Pooling	Web-Mining	0.833	0.909	0.769	0.931	0.01 / 0.6
	Big-Data	0.952	.909	1	1	0.01 / 0.7
	Expert-Manual	0.9	0.818	1	1	0.05 / 0.7
Scheduling	Web-Mining	0.740	0.909	0.625	0.863	0.005 / 0.2
	Big-Data	0.916	1	0.846	0.954	0.001 / 0.2
	Expert-Manual	0.88	1	0.785	0.931	0.01 / 0.4

the medians of paired samples (Note: The data was not normally distributed). Both tests have been recommended for small datasets (as small as 5 per group) [5].

In both statistical tests, we could not reject the null hypothesis (there is a difference in median/mean-rank of two groups.)⁵

Result 1: There is no statistically significant difference between the trace link classification accuracy for a classifier trained using Expert-Based approach and Automated-Web Mining.

⁵ p-value of 0.05

Table 5: Differences in F-Measure of Expert-Based and Manual Approach

Audit	Authenticate	Heartbeat	Pooling	Scheduling
0.048	0	0.209	0.067	0.14

RQ2: Manual Expert-Based Training vs. Automated Big-Data Analysis.

In two of the five tactics, the Big-Data Analysis approach outperformed the manual Expert-Based approach. In two cases both approaches performed very close. Table 6 shows the differences between the f-measure of two approaches.

Table 6: Differences in F-Measure of manual Expert-Based and automated Big-Data Analysis Approach

Audit	Authenticate	Heartbeat	Pooling	Scheduling
0.071	0.356	0.097	-0.052	-0.036

Similar to RQ1, *Wilcoxon* and *Friedman ANOVA* tests were conducted to compare the medians of paired samples. In both cases, the null hypothesis was retained. ⁶

Result 2: There is no statistically significant difference between the trace link classification accuracy for a classifier trained using Expert-Based approach and automated Big-Data Analysis. This indicates that Big-Data Analysis approach can be used as a practical technique to help software traceability researchers generate datasets.

6 Cost-Benefit Analysis

Our empirical study of the tree baseline training set creation techniques suggests that there is no statistically significant differences between trace link accuracy for a classification technique trained using each of these techniques. However the cost of employing experts to help in establishing the training set is significantly higher than automated approaches, while the obtained results are not different.

Cost-Comparison In an earlier work, the estimated cost (in terms of time) for creating the training set using Expert-Based approach for 5 tactics was 1080 hours. Taking into account the hourly salary of an expert or even a student in terms of dollar per hour will make this approach cost thousands of dollars. The automated Big-Mining approach generates similar code snippets dataset within a few seconds.

One drawback for any automated data-mining based approach is the inherent inaccuracy of these techniques. To better investigate this fact in our automated techniques, two members of our team manually evaluated the automatically generated training-sets. The accuracy of each training-set per tactic is

⁶ p-value of 0.05

shown in table 7. Overall, the automated approach based on Big-Data analysis has created more correct data points (code snippets) than the web-mining approach. This might be because of the amount of noise on the technical libraries as well as inaccuracies in the underlying search technique used by the web-mining approach.

Table 7: Quality of automatically generated training-set

	Audit	Scheduling	Authentication	Heartbeat	Pooling
Web-Mining	0.6	1	0.91	0.6	0.8
Big-Data Analysis	1	1	1	0.9	0.9

We also compared the data quality in two baseline methods of Big-Data analysis and Manual method. While in over 90% of cases the Big-Data approach has successfully retrieved correct code snippets from our large scale software repository, we observed that the data collected by experts exhibits higher internal quality. The manually collected training-sets not only contain 100% accurate data points (due the rigorous data collection), the experts have also taken into account the representativeness, diversity, generalizability, as well as quality of these samples for training purposes. The manually collected code-snippets are richer in terms of vocabularies, APIs and comments. Based on our observation, we believe this is one of the main differences in the underlying baseline methods.

Investigating the score assigned to each indicator terms across three baseline techniques, we observed that the indicator terms generated by manually created dataset have bigger probability scores, and are ordered better with less noises (e.g. unrelated terms). In future work, we aim to augment our automated approach so that not only can they find related code-snippets, they will also take into account metrics related to data quality and sampling strategies.

6.1 To What Extent Expert-Based Approach Can be Practical?

An ongoing debate exists on the research techniques examined/developed using students-generated dataset. The community have utilized different mitigation techniques to minimize the biases and threats related to this set of approaches [19, 21]. At the same time, the community have praised the notion of Expert-Based approach in obtaining dataset. Unfortunately there are several threats related to this approach as well, which some of them are similar to student-generated datasets. In this section, we will explore one of these challenges, which is related to the extent such datasets can be useful. **Hypothesis** : It is commonly accepted as a fact that, *the larger the size of training set, the more accurate and generalizable the underling learning method will be*. This is essentially because, when the sample size is large enough, it will more accurately reflect the population it was, and therefore the sample is distributed more closely around the population mean. However, to the best of our knowledge no one has explored whether there is a benefit in extending the training set size generated by the experts. Specially that such extension means can have a significant cost. With all the mitigation techniques used to minimize the threat to validity and create generalizable training set, we do not have scientific confidence in this matter. **Experiment to Investigate:** In the next experiment we investigate the impact of different dataset sizes on accuracy of traceability link discovery. The goal of this experiment is not to prove that the training set size matters or does not matter. instead we aim to perform a cost-benefit analysis for the cases where the training-set is established manually by experts.

For instance, extending the training-set of tactical code snippets from 10 open source projects to 50 projects requires almost 6 additional months of work. The experiment described in this section aims to investigate the increase in accuracy of classifiers for such additional cost.

6.1.1 Experiment Design

In our very first work in this area [30], we used training sets of code snippets from 10 software systems. In extension of this work [25] we used code-snippets sampled through a peer-reviewed process from 50 open source projects. Considering that the training sets were established using systematic manual peer review, it took an extensive amount of time to create such traceability training sets.

Dataset For each of the five tactics included in this study, three experts identified 50 open-source projects in which the tactic was implemented. For each project an architecture biopsy was performed to retrieve the source file in which each utilized tactic was implemented. In addition, for each project a randomly selected non-tactical source files was retrieved.

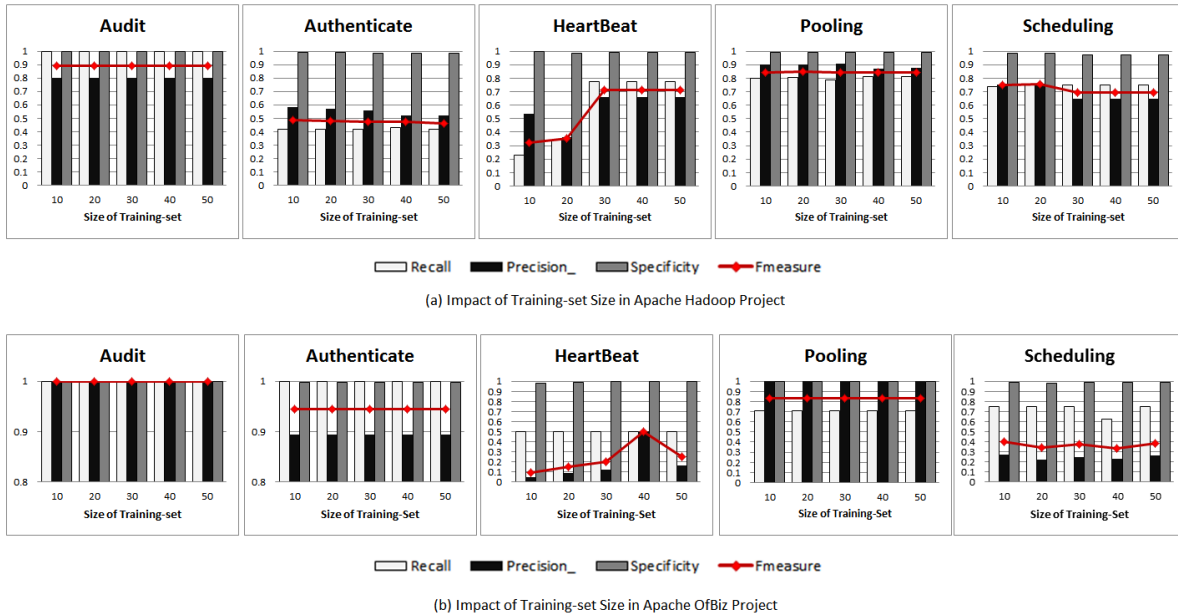


Fig. 4: The impact of training-set size in manually established dataset on accuracy of recovering trace links

Impact of Training set Size on Trace Link Accuracy (Two Case Studies) In this part of research, we investigate RQ3: What is the Impact of Training set Size on the Accuracy of Trace Link Classification? At first the classifiers were trained using 5 sub-samples of this dataset for the size ranges of 10, 20, 30, 40, and 50 sample code snippets. Then each time the classifiers were used against the source code of Apache Hadoop and OfBiz. These two projects are widely used in industry and are representative of complex software systems.

We compare the trace-link accuracy of classifiers trained using different training set sizes. This would help us investigate if there is a return-on-investment for employing experts to establish large(er) training sets.

The accuracy metrics are reported in figure 4. The bars in this graph show precision, recall and specificity [30]. The red line shows the f-measure metric. Except *heartbeat* architectural tactic that exhibits major changes across different training set sizes, in all the other four tactics, the training-set size did not show any significant changes in the accuracy of trained classifier.

Result 3: This observation supports the notion that in case of manually creating a high quality training set, the size of dataset will not have a significant impact on the accuracy of classification technique described in equations 2 and 1. Collecting more data-points by experts will not increase the accuracy or generalizability of the trained classifier.

This observation is only supported by the data obtained from two case studies. In future works, we will run more experiments, to investigate if this would be valid across different systems.

7 Application to the other Area of Requirements Engineering

The previous experiments show the feasibility and practicality of automated training set generation techniques. The results indicate that the Web-Mining and Big-Data Analysis approaches can automatically generate training set with similar quality to expert-created ones.

In this section, we aim to conduct a feasibility study on using the proposed automated dataset creation technique to support research in different areas of requirements engineering. This will provide an answer for RQ4 (stated in section 1).

7.1 Usage Scenario#1: Tracing Regulatory Codes.

This subsection presents the first potential usage scenario for applying the automated dataset generation techniques in the area of tracing regulatory codes. **Problem:** One of the challenges faced by community of researchers in the area of requirements traceability is the lack of datasets such as requirements, implementation or documentations related to regulatory codes within a software domain. There are a limited number of datasets commonly used such as CCHIT or HIPAA which can be found on COEST.ORG website. The proposed research techniques in this area are primarily evaluated by running experiments over sections of the same dataset, or by tracing one of these to the source code of two open source software systems of WorldVista and Itrust.

Feasibility Study: Technical libraries such as MSDN have several documentations, technical guidelines, best practices and preselected technologies and APIs which can be used to address a wide range of regulatory codes, such as HIPAA and SOX [9]. For example, in table 8 we list a set of regulatory-compliance acts which we found significant technical discussions about them on MSDN library.

In a preliminary study, we used our automated technique to create a dataset for the domain of “Tracing Regulatory Code”. We ran a sample experiment to create a dataset for technologies which can be used to address HIPAA regulations related to Database and Security. We evaluated the accuracy of the extracted data, the results are presented in table 9 and indicated that 63% of automatically generated data points were correct. Due to the lack of space we only provide an excerpt of two sample data points:

- “HIPAA compliance: Healthcare customers and Independent Software Vendors (ISVs) might choose SQL Server in Azure Virtual Machines instead of Azure SQL Database because SQL Server in an Azure Virtual Machine is covered by HIPAA Business Associate Agreement (BAA). For information on compliance, see Azure Trust Center.”
- “Confidentiality: Do not rely on custom or untrusted encryption routines. Use OS platform provided cryptographic APIs, because they have been thoroughly inspected and tested rigorously. Use an asymmetric algorithm such as RSA when it is not possible to safely share a secret between the party encrypting and the party decrypting the data....”

7.2 Usage Scenario#2: Classifying Functional Requirements:

Another area where automated techniques can be used is generating datasets for the problem of classifying/tracing functional requirements.

Problem: Traditionally VSM [18] technique has been widely used to trace functional requirements to source code. On the other hand, there are studies showing the feasibility of using supervised learning methods to trace reoccurring functional requirements [6]. The biggest drawback for this approach is the difficulty of obtaining several samples of the same functional requirements or code snippets.

Feasibility Study: Using Big-Data analysis we observed that, in our ultra-large code repository, there are a large number of software systems from the same domain, therefore the code snippets to implement functional requirements will also reoccur across these systems. Therefore it is possible to collect datasets of such implementation and use different supervised learning techniques to detect these types of requirements in the source code or utilize such dataset for other purposes. Table 9 shows the accuracy of the Big-Data analysis in establishing datasets for code snippets implementing functional requirements of

Table 8: Sample Regulations Discussed on Technical Libraries

ACT Name	Aplies to
Sarbanes Oxley Act	Legislation passed by the U.S. Congress to protect shareholders and the general public from accounting errors and fraudulent practices in the enterprise, as well as improve the accuracy of corporate disclosures [9]. More on (http://www.sec.gov/)
HIPAA	the federal Health Insurance Portability and Accountability Act of 1996. The primary goal of the law is to make it easier for people to keep health insurance, protect the confidentiality and security of health care information and help the health care industry control administrative costs. [9]
PCI	Payment Card Industry Data Security Standard(PCI DSS) is a proprietary information security regulation for organizations that handle branded credit cards. [4]
The Gramm-LeachBliley Act (GLBA)	Also known as the Financial Services Modernization Act of 1999, is an act of the 106th United States Congress, removing barriers for confidentiality and integrity of personal financial information stored by financial institutions. [2]
SB 1386	California S.B. 1386 was a bill passed by the California legislature. The first of many U.S. and international security breach notification laws. Enactment of a requirement for notification to any resident of California whose unencrypted personal information was, or is reasonably believed to have been, acquired by an unauthorized person. [1]
BASEL II	Is recommendations on banking laws and regulations issued by the Basel Committee on Banking Supervision. Aplies to: Confidentiality and integrity of personal financial information stored by financial institutions. Availability of financial systems. Integrity of financial information as it is transmitted. Authentication and integrity of financial transactions [9].
Health Level Seven (HL7)	Provides regulations for the exchange of data among health care computer applications that eliminate or substantially reduce the custom interface programming and program maintenance that may otherwise be required [9].

an ERP (Enterprise Resource Planning) software system. In fact, in all cases, our approach successfully created datasets of code snippets to implement those requirements. The terms in the queries to retrieve the implementation of functional requirements were directly extracted from an on-line document of a similar system ⁷.

8 Tool Support

A functional prototype of the automated approaches is developed and released as a web-based tool called BUDGET (Bigdata aUgmented Dataset GEneration Tool)⁸. BUDGET's inputs are the name of the tactic of interest, which approach(es) to use when collecting the data - *Web Mining* or *Big-Data Analysis* - and the dataset size to be generated. Furthermore, there are more advanced sampling parameters that can be tuned if a particular data sampling strategy needs to be followed. This becomes especially useful for Big-Data analysis approach where the user has access to index source code of more than 100,000 applications. The sampling gives the user the flexibility to retrieve the tactical implementations from a single project, many projects, or the entire repository.

Figure 5 shows the user interface for specifying generic parameters of the BUDGET tool. As shown in this figure, the generated dataset size can be either balanced (equal amount of negative and positive samples generated) or unbalanced (different sizes of positive and negative samples) and the datasets can be automatically created using both Web Mining and Big Data analysis techniques or only one.

When the *Web Mining* approach is chosen, BUDGET will collect a set of web pages related to a tactic selected from technical libraries. The user can specify the list of technical libraries in a comma-separated list of URLs. By default, the tool uses MSDN as an information source if no other libraries are provided. Figure 6 shows the form field for indicating the technical libraries.

In order to retrieve tactical-related web pages, BUDGET uses Google Search Engine APIs to query technical programming libraries. Tactical terms collected from textbooks are used as a search query. Then BUDGET creates positive/tactical samples by extracting the content of web pages in top search results (i.e. HTML tags are filtered out). A similar process is followed to generate negative samples; the only difference is that the search query is modified to only return web pages that do not contain any of the tactic-related terms.

⁷ Please see terms in the figures: http://www.1tech.eu/clients/casestudy_ventraq

⁸ <http://design.se.rit.edu/budget/>

Table 9: Accuracy of automatically generated datasets in two different areas of requirements engineering

Approach	Query	Correct
Big-Data	Query 1: Billing, Bill Calculation, Invoice Generation	90%
	Query 2: Balance Management, Credit Management, Account Management, Credit Card Processing	100%
	Query 3: Business Intelligence, SLA Management, Database Marketing	100%
	Query 4: Product Shipment, Shopping	100%
Web-Mining	Database Security HIPAA	63%

Fig. 5: User interface for selecting the data generation parameters of BUDGET tool

(a) Configuration parameters for Big Data Analysis

(b) UI for specifying the technical libraries for Web Mining

(a) Output of the tool, downloaded as a zip file

Fig. 6: Configuration parameters for Big Data Analysis, Web-Mining Approach and Generated Output

When using the *Big Data Analysis* technique, BUDGET will retrieve source code files from public code repositories to generate the datasets. Currently BUDGET's source code repository contains over 116,000 projects, continuously more open source projects are being added to this repository.

BUDGET's parameter for generating training set of tactical code snippets include programming languages of the source codes, a sampling strategy (Figure 6). The sampling strategy defines how BUDGET should sample the tactic-related code snippets from the our ultra-large scale repositories. The three possible sampling strategies are: *Best Cases*, *Random Sampling* and *Stratified Sampling*. These strategies work as follows:

- *Best Cases*: In this strategy, the tactical files with the highest similarity score are returned. By default, the entire source code repository is used for drawing the samples, unless the user specifies a list of repositories to limit the sampling.
- *Random Sampling*: In this strategy, first the user specifies the sampling population by defining the percentage of tactical files to be included in the base population. BUDGET first separates top P % of tactical files (P defined by the user), then randomly generates a dataset size of N (where N is defined by the user).
- *Stratified Sampling*: For each project in the repository (or user defined list), only X tactical source code files are randomly selected. The value of X is also indicated by the user.

After selecting the sampling strategy, the sampled tactical files are sorted based on the similarity score to the tactic query. Subsequently, the tool generates the N positive and M negative samples defined by the user. For that, the tool selects the N most similar tactical files and the M least related files for generating the positive and negative samples, respectively.

Besides using the tactical terms for the Big-Data Analysis and Web Mining approaches, the tool has the flexibility of using user-defined terms to generate datasets. In this situation, instead of using our own set of tactical terms, BUDGET applies the terms specified by the user in the *Web Mining* and *Big-Data Analysis* techniques.

After the datasets are generated, the BUDGET makes them available for download as a compressed file in ZIP format. This ZIP file will contain two folders: one has textual files obtained from Web Mining and the other has source code files generated from Big-Data Analysis. Each folder has two subdirectories for separating positive from negative cases. Figure 6 shows the folder hierarchy of the datasets generated.

9 Interpretation of Results and Reproducibility

Extrapolating the results of empirical studies beyond the context of the experiments and the data used in them can be risky. Our empirical study is not an exception. We compared three baseline data generation techniques. The results indicate that automated data generation techniques resulted in the same trace link accuracy as expert-based approach. This conclusion was drawn based on study of five architectural tactics. However our further experiments have shown that several tactics share similar characteristics at the code level, and can be detected using text analysis. Therefore, it is possible to utilize BUDGET for automating generation of training set for a large number of architectural tactics. We expect to observe different accuracy.

The impact of dataset size in case of expert-created training set was evaluated using two industrial case study. Although these are large scale, representative industrial projects, we believe further experiments would be beneficial to support/disprove our observation.

Since BUDGET is accessible for the public, it would enable the researchers in the community to conduct similar experiments, reproduce the results and expand this work. The expert-based dataset used in investigating the impact of dataset size is also released on-line at COEST.org.

10 Threats To Validity

Threats to validity can be classified as *construct*, *internal*, *external*, and *statistical* validity. We discuss the threats which potentially impacted our work, and the ways in which we attempted to mitigate them.

External validity evaluates the generalizability of the approach. One of the primary threats is related to the construction of the datasets for this study. The manual dataset included over 250 samples of tactic-related code. The task of locating and retrieving these code snippets was conducted primarily by two experts in the area of requirements and software architecture and was reviewed by two additional experts. This was a very time-consuming task that was completed over the course of three months. The systematic process we followed to find tactic related classes and the careful peer-review process gave us confidence that each of the identified code snippets was indeed representative of its relevant tactic. In addition, all of the experiments conducted in our study were based on Java, C# and C code. Some of the identified keyterms are influenced by the constructs in these programming languages such as calls to APIs that support specific tactic implementation. Furthermore, the Hadoop and OfBiz case studies were designed to evaluate the impact of dataset size on accuracy of tactic classification on a large and realistic system. We therefore expect them to be representative of a typical software engineering environment, which suggests that it could generalize to a broader set of systems. On the other hand, the majority of identified keyterms are non-language specific. The experimental results reported in this paper will not be impacted by this issue.

Construct validity evaluates the degree to which the claims were correctly measured. The n-fold cross-validation experiments we conducted are a standard approach for evaluating results when it is difficult to gather larger amounts of data. To avoid the impact of dataset size on training-set quality, all the comparison experiments were conducted on the training set of equal size.

Internal validity reflects the extent to which a study minimizes systematic error or bias, so that a causal conclusion can be drawn. A greater threat to validity is that the search for specific tactics was limited by the preconceived notions of the researchers, and that additional undiscovered tactics existed that used entirely different terminology. However we partially mitigated this risk through locating tactics using searching, browsing, and expert opinion. Since multiple data collection mechanisms were used by experts, the dataset is not dependent on a limited number of terms, in fact in some cases there a large diversity in terminologies. In the case of the Hadoop project, we elicited feedback from Hadoop developers on the open discussion forum. Another threat in this category is that, the accuracy automated techniques can be dependent to the search query used in the study. To avoid this bias, we preselected the queries from description of tactics from text book. In future work we are planning to run different experiments to identify the impact of domain knowledge of the person who creates the query on the quality of datasets.

Statistical validity concerns whether the statistical analysis has been conducted correctly. In order to address this threat appropriate statistical techniques were used. For reliability of conclusions we used two non-parametric tests. Uniformly both tests indicated that there is no statistically differences between the accuracy of training methods although manual ones ranks the best.

11 Conclusion

In this paper we presented three baseline techniques for creating training-set to train a classifier to detect architectural tactics and establish traceability links form architecturally significant requirements to tactics and source code. Our analysis shows that automated techniques can generate useful training-sets with mostly similar quality to a expert-created datasets. The proposed automation techniques can be used in the area of tracing architecturally significant requirements as well as other software engineering areas. The long-term goal of this project is to develop automated techniques capable of creating scientific dataset with similar/or better quality of the expert create datasets. In future work, we will investigate the practicality of the automated techniques in other domains. We will also extend our work to provide

more fine-grained sampling of the source files or web-pages and therefore reduce the potential noise in the final training sets. We also plan to run more experiments to understand the impact of trace user's domain knowledge on quality of established datasets.

References

1. California Senate Bill SB 1386, Sept. 2002. http://www.leginfo.ca.gov/pub/13-14/bill/sen/sb_1351-1400/sb_1351_bill_20140221_introduced.pdf.
2. US Congress. Gramm-Leach-Bliley Act, Financial Privacy Rule. 15 USC 6801–6809, November 1999. http://www.law.cornell.edu/uscode/usc_sup_01_15_10_94_20_I.html.
3. Koders.
4. P. C. I. Council. Payment card industry (pci) data security standard. Available over the Internet - July 2010. <https://www.pcisecuritystandards.org>.
5. Using the Student's t-test with extremely small sample sizes.
6. P. R. Anish, B. Balasubramaniam, J. Cleland-Huang, R. Wieringa, M. Daneva, and S. Ghaisas. Identifying architecturally significant functional requirements. In *Proceedings of the Fifth International Workshop on Twin Peaks of Requirements and Architecture*, TwinPeaks '15, pages 3–8, Piscataway, NJ, USA, 2015. IEEE Press.
7. F. Bachmann, L. Bass, and M. Klein. *Deriving Architectural Tactics: A Step Toward Methodical Architectural Design*. Technical Report, Software Engineering Institute, 2003.
8. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison Wesley, 2003.
9. G. W. Beeler, Jr. and D. Gardner. A requirements primer. *Queue*, 4(7):22–26, Sept. 2006.
10. C. E. Brodley. Addressing the selective superiority problem: Automatic algorithm/model class selection, 1993.
11. J. R. Cano, F. Herrera, and M. Lozano. Using evolutionary algorithms as instance selection for data reduction in kdd: An experimental study. *Trans. Evol. Comp.*, 7(6):561–575, Dec. 2003.
12. J. Cleland-Huang, B. Berenbach, S. Clark, R. Settini, and E. Romanova. Best practices for automated traceability. *Computer*, 40(6):27–35, 2007.
13. J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker. A machine learning approach for tracing regulatory codes to product specific requirements. In *ICSE (1)*, pages 155–164, 2010.
14. J. Cleland-Huang, O. Gotel, J. Huffman Hayes, P. Mader, and A. Zisman. Software traceability: Trends and future directions. In *Proc. of the 36th International Conference on Software Engineering (ICSE), India*, 2014.
15. J. Cleland-Huang, R. Settini, X. Zou, and P. Solc. Automated detection and classification of non-functional requirements. *Requir. Eng.*, 12(2):103–120, 2007.
16. R. Dyer, H. Rajan, H. A. Nguyen, and T. N. Nguyen. Mining billions of ast nodes to study actual and potential usage of java language features. In *Proceedings of the 36th International Conference on Software Engineering, ICSE 2014*, pages 779–790, New York, NY, USA, 2014. ACM.
17. G. Gates. The reduced nearest neighbor rule (corresp.). *Information Theory, IEEE Transactions on*, 18(3):431–433, May 1972.
18. M. Gethers, R. Oliveto, D. Poshyvanyk, and A. Lucia. On integrating orthogonal information retrieval methods to improve traceability recovery. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 133–142, Sept 2011.
19. M. Gibiec, A. Czauderna, and J. Cleland-Huang. Towards mining replacement queries for hard-to-retrieve traces. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, ASE '10*, pages 245–254, New York, NY, USA, 2010. ACM.
20. R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. Morgan Kaufmann, 1995.
21. G. A. Liebchen and M. Shepperd. Data sets and data quality in software engineering. In *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering, PROMISE '08*, pages 39–44, New York, NY, USA, 2008. ACM.
22. A. Mahmoud. An information theoretic approach for extracting and tracing non-functional requirements. In *Proc. RE*, pages 36–45. IEEE, 2015.
23. M. McCandless, E. Hatcher, and O. Gospodnetic. *Lucene in Action, Second Edition: Covers Apache Lucene 3.0*. Manning Publications Co., Greenwich, CT, USA, 2010.
24. J. C.-H. Mehdi Mirakhorli. Detecting, tracing, and monitoring architectural tactics in code. *IEEE Trans. Software Eng.*, 2015.
25. M. Mirakhorli. Preserving the quality of architectural decisions in source code, PhD Dissertation, DePaul University Library, 2014.

26. M. Mirakhorli and J. Cleland-Huang. *Tracing Non-Functional Requirements*. In: Andrea Zisman, Jane Cleland-Huang and Olly Gotel. *Software and Systems Traceability*, Springer-Verlag., 2011.
27. M. Mirakhorli and J. Cleland-Huang. Using tactic traceability information models to reduce the risk of architectural degradation during system maintenance. In *Proceedings of the 2011 27th IEEE International Conference on Software Maintenance, ICSM '11*, pages 123–132, Washington, DC, USA, 2011. IEEE Computer Society.
28. M. Mirakhorli, A. Fakhry, A. Grechko, M. Wieloch, and J. Cleland-Huang. Archie: A tool for detecting, monitoring, and preserving architecturally significant code. In *CM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2014)*, 2014.
29. M. Mirakhorli, P. Mäder, and J. Cleland-Huang. Variability points and design pattern usage in architectural tactics. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, FSE '12*, pages 52:1–52:11. ACM, 2012.
30. M. Mirakhorli, Y. Shin, J. Cleland-Huang, and M. Cinar. A tactic centric approach for automating traceability of quality concerns. In *International Conference on Software Engineering, ICSE (1)*, 2012.
31. M. L. C. Passini, K. B. Estbanez, G. P. Figueredo, and N. F. F. Ebecken. A strategy for training set selection in text classification problems. (*IJACSA*) *International Journal of Advanced Computer Science and Applications*, 4(6):54–60, 2013.
32. G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.
33. D. B. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 293–301. Morgan Kaufmann, 1994.
34. I. University of California. The sourcerer project. sourcerer.ics.uci.edu.
35. D. R. Wilson and T. R. Martinez. Reduction techniques for instance-based learning algorithms. *Mach. Learn.*, 38(3):257–286, Mar. 2000.
36. J. Zhu, M. Zhou, and A. Mockus. Patterns of folder use and project popularity: A case study of github repositories. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '14*, pages 30:1–30:4, 2014.