

Rochester Institute of Technology

**RIT Digital Institutional Repository**

---

Theses

---

12-17-2019

## **Towards an Intelligent System for Software Traceability Datasets Generation**

Waleed Abdu Zogaan  
waz7355@rit.edu

Follow this and additional works at: <https://repository.rit.edu/theses>

---

### **Recommended Citation**

Zogaan, Waleed Abdu, "Towards an Intelligent System for Software Traceability Datasets Generation" (2019). Thesis. Rochester Institute of Technology. Accessed from

This Dissertation is brought to you for free and open access by the RIT Libraries. For more information, please contact [repository@rit.edu](mailto:repository@rit.edu).

# **Towards an Intelligent System for Software Traceability Datasets Generation**

by

**Waleed Abdu Zogaan**

A dissertation submitted in partial fulfillment of the  
requirements for the degree of

**Doctor of Philosophy  
in Computing and Information Sciences**

B. Thomas Golisano College of Computing and  
Information Sciences

Rochester Institute of Technology

Rochester, New York

December 17<sup>th</sup> 2019

# Towards an Intelligent System for Software Traceability Datasets Generation

by

Waleed Abdu Zogaan

**Committee Approval:**

We, the undersigned committee members, certify that we have advised and/or supervised the candidate on the work described in this dissertation. We further certify that we have reviewed the dissertation manuscript and approve it in partial fulfillment of the requirements of the degree of Doctor of Philosophy in Computing and Information Sciences.

---

Dr. Mehdi Mirakhorli  
Dissertation Advisor

Date

---

Dr. Venera Arnaoudova  
Dissertation Committee Member

Date

---

Dr. Christian Newman  
Dissertation Committee Member

Date

---

Dr. Mohamed Mkaouer  
Dissertation Committee Member

Date

---

Dr. Daniel Phillips  
Dissertation Defense Chairperson

Date

**Certified by:**

---

Dr. Pengcheng Shi  
Ph.D. Program Director, Computing and Information Sciences

Date



# Towards an Intelligent System for Software Traceability Datasets Generation

by

**Waleed Abdu Zogaan**

Submitted to the

B. Thomas Golisano College of Computing and Information Sciences

Ph.D. Program in Computing and Information Sciences

in partial fulfillment of the requirements for the

**Doctor of Philosophy Degree**

at the Rochester Institute of Technology

## Abstract

Software datasets and artifacts play a crucial role in advancing automated software traceability research. They can be used by researchers in different ways to develop or validate new automated approaches. Software artifacts, other than source code and issue tracking entities, can also provide a great deal of insight into a software system and facilitate knowledge sharing and information reuse. The diversity and quality of the datasets and artifacts within a research community have a significant impact on the accuracy, generalizability, and reproducibility of the results and consequently on the usefulness and practicality of the techniques under study. Collecting and assessing the quality of such datasets are not trivial tasks and have been reported as an obstacle by many researchers in the domain of software engineering. In this dissertation, we report our empirical work that aims to automatically generate and assess the quality of such datasets. Our goal is to introduce an intelligent system that can help researchers in the domain of software traceability in obtaining high-quality “training sets”, “testing sets” or appropriate “case studies” from open source repositories based on their needs. In the first project, we present a first-of-its-kind study to review and assess the datasets that have been used in software traceability research over the last fifteen years. It presents and articulates the current status of these datasets, their characteristics, and their threats to validity. Second, this dissertation introduces a Traceability-Dataset Quality Assessment (T-DQA) framework to categorize software traceability datasets and assist researchers to select appropriate datasets for their research based on different characteristics of the datasets and the context in which those datasets will be used. Third, we present the results of an empirical study with limited scope to generate datasets using three baseline approaches for the creation of training data. These approaches are (i) 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. Finally,

we conduct a large-scale study to identify which types of software artifacts are produced by a wide variety of open-source projects at different levels of granularity. Then we propose an automated approach based on Machine Learning techniques to identify various types of software artifacts. Through a set of experiments, we report and compare the performance of these algorithms when applied to software artifacts. Finally, we conducted a study to understand how software traceability experts and practitioners evaluate the quality of their datasets. In addition, we aim at gathering experts' opinions on all quality attributes and metrics proposed by T-DQA.

## Acknowledgments

First and foremost I would like to thank my advisor Prof.Mehdi Mirakhorli, for his endless guidance, encouragement, and patience. Since the beginning of my journey as a graduate during my master degree and over the years during my Ph.D.studies he has become not just an advisor, but a good friend. All the credit goes to him for the researcher and academic I have become today.

I would like also to thank my committee members Venera Arnaoudova, Mohamed Mkaouer, and Christian Newman for serving on my dissertation committee and for all of the discussions that have helped me in my research and to improve this dissertation.

Thank you to Pengcheng Shi for all of his valuable advice, and for helping to guide me through my graduate studies.

*To the soul of my beloved father Abdu who supported me and whom I wish he is here today to share this moment with me.*

*To my mother, Najat, for her endless love, prays and encouragement.*

*To my wife, Hend, whose love and confidence is a constant source of inspiration. She is the star I follow to find my path in the dark nights.*

*To my siblings, Wail, Wesam, Reham and Rawan for everything they have done for me, and for always being there for me.*

*And to my kids, Emaa, Khalid, Abdu, Mohab and Wafaa who were my companions in this long journey and whose smiles brighten up my life.*



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Research Goals . . . . .	2
1.2	List of Contributions . . . . .	4
1.3	List of Publications . . . . .	5
1.4	Dissertation Organization . . . . .	6
<b>2</b>	<b>Background</b>	<b>7</b>
2.1	SLRs in Traceability . . . . .	7
2.2	Open-Source Software as a Dataset . . . . .	9
2.3	Assessing the Quality of Datasets . . . . .	9
2.4	Automated Datasets Generation . . . . .	11
2.5	Categorization of Software Artifacts . . . . .	11
2.6	Surveys in software engineering . . . . .	12
<b>3</b>	<b>Methodology</b>	<b>14</b>
3.1	Research agenda . . . . .	14
3.2	Goal 1: Investigate the characteristics and types of software traceability datasets: . . . . .	16
3.2.1	Research questions . . . . .	16
3.2.2	Search strategy . . . . .	18
3.2.3	Inclusion and exclusion criteria . . . . .	19
3.2.4	Study selection process . . . . .	19
3.2.5	Data extraction . . . . .	21
3.3	Goal 2: Building data quality framework: . . . . .	21
3.4	Goal 3: Feasibility of automatically generate datasets from open source software repositories: . . . . .	21
3.4.1	Research questions . . . . .	22
3.4.2	Study scope . . . . .	22
3.4.3	Traceability challenge: identifying Tactic-Related classes . . . . .	23
3.4.4	Overview of the three baseline techniques . . . . .	24
3.4.5	Baseline method 1: expert-created approach . . . . .	25
3.4.6	Web-mining approach . . . . .	27
3.4.7	Big-Data analysis approach . . . . .	29
3.4.8	Experiment overview . . . . .	31
3.4.9	Experiment design . . . . .	32

3.4.10	Evaluation metrics . . . . .	33
3.4.11	Minimizing biases . . . . .	33
3.5	Goal 4: Classification, automated categorization, and detection of open-source software artifacts: . . . . .	34
3.5.1	Research questions . . . . .	34
3.5.2	Study definition and design . . . . .	34
3.5.3	Subject Systems . . . . .	35
3.5.4	Oracle . . . . .	36
3.5.5	Automatic Artifact Classification . . . . .	36
3.5.6	Evaluation . . . . .	38
3.6	Goal 5: Traceability datasets quality assessment survey: . . . . .	39
3.6.1	Research questions . . . . .	39
3.6.2	Survey Design . . . . .	39
3.6.3	Participants . . . . .	40
3.6.4	Pilot Study . . . . .	40
3.6.5	Data Collection . . . . .	40
3.6.6	Analysis . . . . .	40
3.7	Goal 6: T-DQA Web-Tool: . . . . .	41
3.7.1	Datasets collection . . . . .	41
3.7.2	T-DQA metrics . . . . .	42
<b>4</b>	<b>Results</b> . . . . .	<b>46</b>
4.1	RQ1: What are the characteristics of traceability datasets? . . . . .	46
4.1.1	<i>RQ1.1: What are the source and target artifacts in traceability datasets?</i> . . . . .	46
4.1.2	<i>RQ1.2: Which application domains are represented by traceability datasets?</i> . . . . .	47
4.1.3	<i>RQ1.3: What is the size of traceability datasets?</i> . . . . .	49
4.1.4	<i>RQ1.4: What proportion of the traceability datasets is from industry, open-source projects, and student generated data?</i> . . . . .	49
4.1.5	<i>RQ1.5: Are traceability datasets available for reuse?</i> . . . . .	50
4.1.6	<i>RQ1.6: Is there a relation between the characteristics and the quality of traceability datasets on the one hand and their reusability on the other hand?</i> . . . . .	50
4.1.7	<i>RQ1.7: What are the threats to validity associated with traceability datasets?</i> . . . . .	50
4.1.8	<i>RQ1.8: Do we, as a community, strive for a diversity of traceability datasets?</i> . . . . .	52
4.2	RQ2: How to assess the quality of traceability datasets? . . . . .	53
4.3	RQ3: Is it feasible to automatically generate datasets from open source software repositories? . . . . .	55
4.3.1	<i>RQ3.1: Does the training method based on automated web-mining result in higher trace-links classification accuracy compared to an expert-created training set?</i> . . . . .	55
4.3.2	<i>RQ3.2: Does the training method based on automated big-data result in higher trace-links classification accuracy compared to an expert-created training set?</i> . . . . .	59
4.3.3	<i>RQ3.3: What is the impact of training set size on the accuracy of trace link classification?</i> . . . . .	59
4.3.4	Cost-Benefit analysis . . . . .	61
4.3.5	Tool support . . . . .	62

4.3.6	Discussions . . . . .	64
4.3.7	Generalization of results to other classification techniques . . . . .	65
4.3.8	Qualitative insights . . . . .	66
4.3.9	Application to the other Areas of Requirements Engineering . . . . .	68
4.3.10	Usage Scenario#2: Classifying Functional Requirements: . . . . .	70
4.3.11	Threats To Validity . . . . .	70
4.4	RQ4: Can we automatically detect and categorize open-source software artifacts? . . . . .	71
4.4.1	RQ4.1: How can software artifacts be categorized? . . . . .	72
4.4.2	RQ4.2: How accurate is the proposed approach for automatic software artifact classification? . . . . .	76
4.5	RQ5: What types of artifacts are created during open-source software development? . . . . .	79
4.6	RQ6: How do experts assess the quality of traceability datasets? . . . . .	81
4.6.1	RQ6.1: What are the quality attributes that researchers are looking for when they select datasets? . . . . .	81
4.6.2	RQ6.2: What dataset qualities have an impact on the meaningful conclusions being drawn from a research project? . . . . .	86
4.6.3	RQ6.3: What are the datasets quality-attributes that could impact the generalizability of research results? . . . . .	86
4.7	RQ7: Does the existing framework for evaluating the quality of traceability datasets captures the relevant characteristics that experts are looking for? . . . . .	88
4.7.1	Discussion . . . . .	90
4.8	T-DQA Web-tool support . . . . .	92
<b>5</b>	<b>Conclusions</b>	<b>95</b>
	<b>Appendices</b>	<b>110</b>
<b>A</b>	<b>Traceability Datasets Quality Survey</b>	<b>111</b>

# List of Figures

3.1	Search process stages. . . . .	20
3.2	Overview of Automated Approaches to Create Tactic Traceability Training-sets	25
3.3	Two sample API descriptions from technical libraries of (a) MSDN and (b) Oracle . . . . .	28
3.4	Approach overview. . . . .	35
4.1	Common <i>Source</i> and <i>Target</i> Artifacts. . . . .	47
4.2	Dataset Domains and Frequency of Use. . . . .	48
4.3	Source and availability of datasets. . . . .	49
4.4	Random Forest Importance Variable Plot. . . . .	51
4.5	Authors and their Dataset Diversity. . . . .	53
4.6	Results for Detection of Tactic-related Classes at various Classification and Term Thresholds for five Different Tactics . . . . .	57
4.7	The impact of training-set size in manually established dataset on accuracy of recovering trace links . . . . .	61
4.8	User interface for selecting the data generation parameters of BUDGET tool	63
4.9	Configuration parameters for Big Data Analysis, Web-Mining Approach and Generated Output . . . . .	63
4.10	Distribution of primary languages in the sampled projects. . . . .	72
4.11	Demographic information about the participants. . . . .	81
4.12	Examples of Excerpt from online survey responses, identified codes in open coding activities, and emerged categories. . . . .	82
4.13	Codes emerged from open coding activities and their frequencies. . . . .	83
4.14	Quality attributes used by the experts to select datasets and their frequencies in the experts' responses. . . . .	84
4.15	Expert's opinion on the relationship between the dataset qualities and research conclusions. . . . .	87
4.16	Expert's opinion on the relationship between the dataset qualities and generalizability of the results. . . . .	87
4.17	Importance of T-DQA attributes from participant's perspective. . . . .	89
4.18	T-DQA Web-Tool Interface . . . . .	92
4.19	Datasets Details . . . . .	93
4.20	Dataset Search Example . . . . .	94

# List of Tables

3.1	Venues used in the manual search phase. . . . .	18
3.2	Databases used in the automatic search phase. . . . .	19
3.3	Inclusion and exclusion criteria. . . . .	19
3.4	Extracted dataset items. . . . .	20
3.5	Manual Dataset Generated by Experts . . . . .	26
3.6	Overview of the projects in Source Code Repository of Big-Data Analysis Approach . . . . .	30
3.7	Existing Traceability-Datasets Sources . . . . .	42
3.8	T-DQA Web-Tool Traceability-Datasets Summary . . . . .	43
3.9	List of T-DQA Web-Tool Datasets . . . . .	44
3.10	T-DQA Web-Tool Metrics . . . . .	45
4.1	Datasets' trace space statistics. . . . .	49
4.2	Traceability-Dataset Quality Assessment (T-DQA) Framework: dimensions, and definitions. . . . .	54
4.3	Indicator terms learned during training . . . . .	56
4.4	A Summary of the Highest Scoring Results . . . . .	58
4.5	Differences in F-Measure of the Expert-Created and Web-Mining Approaches . . . . .	58
4.6	Differences in F-Measure of manual expert-created and automated Big-Data Analysis Approach . . . . .	59
4.7	Accuracy of automatically generated training-set . . . . .	62
4.8	Differences in F-Measure of Expert-Created and Big-Data Analysis Approach in Naïve Bayes Approach . . . . .	65
4.9	Differences in F-Measure of Expert-Created and Web-Mining Approach in Naïve Bayes Approach . . . . .	66
4.10	Sample Regulations Discussed on Technical Libraries . . . . .	69
4.11	Accuracy of automatically generated datasets in two different areas of requirements engineering . . . . .	73
4.12	Heuristics applied to identify types of non-documentation related artifacts. . . . .	73
4.13	Extension distribution in the sampled projects. . . . .	75
4.14	Sample list of features. . . . .	76
4.15	Performance of individual classifiers and 10-fold cross-validation on the training dataset. . . . .	77
4.16	Performance of the classifiers using ensemble learning and 10-fold cross-validation on the <u>training dataset</u> . . . . .	77

4.17 Performance of the classifiers using ensemble learning and 10-fold cross-validation on the testing dataset. . . . .	78
4.18 Distribution of the different types of software artifacts in the sampled projects.	80
4.19 Updated Framework: <i>T-DQA v.2</i> . . . . .	91

# Chapter 1

## Introduction

The requirements of any software, both functional (FR) and non-functional (NFR), come in different formats and sizes. These requirements represent the user's desires for the software functional features and quality constraints. The ability to trace software requirements to its code-related artifacts is beneficial to all stakeholders. In fact, for safety-critical systems, having such trace links is essential to verify the software compliance with the quality requirements. Extracting such trace links manually is time-consuming, expensive and not a trivial task. Therefore, different automated techniques were introduced by researchers to automatically generate software trace links with less effort and cost while achieving high quality.

Automated traceability techniques that rely on machine learning (ML) and information retrieval (IR) are widely and increasingly being used by researchers and developers. Advances in this area of research relies on the availability of different types of *datasets* and *artifacts*. *Training sets* are needed to train trace-algorithms based on ML techniques. For instance, researchers have used datasets of functional requirements and non-functional requirements to train classification techniques to create traceability links between quality attributes and requirements document, design models and source code [46, 127, 144, 151, 167]. *Validation sets* are needed to tune algorithm parameters of such trace-algorithms [40, 108, 127]. *Testing sets* are used to test the performance of trace-algorithms on unseen data. For instance, researchers have used datasets to evaluate the accuracy of a trace-algorithms based on IR techniques to establish links between requirements and source code [57, 63, 82, 167, 177].

Obtaining such software development datasets and manually identify the types of artifacts available or lacking in a specific open-source project has been one of the most frequently reported barriers for researchers in the software engineering domain in general [106, 159]. This problem is even more acute in the area of requirements traceability which is crucial in safety-critical and highly regulated application domains [42]. Many of the publicly available open-source systems are not representative of those domains and consequently may not be suitable to use for training, validation, or testing sets. Thus, a key challenge is to determine the quality of datasets used to conduct a study, to develop an automated technique, or to validate the results of research work. Furthermore, it is important to evaluate the quality of the datasets and to explicitly state the threats to validity associated with the datasets so that research results are articulated with perspective to the underlying datasets.

Despite the crucial role of trace datasets, few efforts have been taken to understand the characteristics and limitations of the datasets in the area of requirements traceability [85] as well as the threats to validity associated with the results obtained with these datasets. Similarly, few efforts have been taken to standardize how the datasets quality tracking and assurance should be implemented [161]. To the best of our knowledge, no previous work assesses or defines quality dimensions and metrics for traceability datasets.

These challenges motivated us to investigate the possibility of using automated big-data analysis approaches to generate scientific datasets from thousands of open source projects. Our overall goal in this research project is to introduce an *intelligent system for software traceability datasets generation* that can help researchers in the domain of software traceability in obtaining high-quality “training sets”, “testing sets” or appropriate “case studies” from open source repositories based on their needs.

## 1.1 Research Goals

Throughout this research project we aim to achieve the following research goals:

- **Goal<sub>1</sub>.** *Investigate the statues, characteristics of the existing software traceability datasets that have been used by the researchers in the community.*

This study will shed some light on the current status of traceability datasets which have been used by researchers and reveals their different characteristics. This will provides us with solid ground and understanding of the characteristics and limitations of the datasets in the area of requirements traceability. This knowledge will play a crucial role when building our datasets generation tool. In addition, it will provide our community with the ability to detect the areas that can improve the rigoroussness of evaluation and practicality of research.

- **Goal<sub>2</sub>.** *Assess the quality of traceability datasets.*

The adoption of a traceability-datasets quality framework will help researchers evaluate the quality of the datasets and their relevancy for specific research tasks. Furthermore, the results and analysis of this study will provide an insight into the tacit community-wide threats related to the datasets which will help to detect both the strengths and weaknesses of our empirical foundations.

- **Goal<sub>3</sub>.** *Automatically generate datasets from open source software repositories.*

Automated traceability techniques that rely on machine learning (ML) and information retrieval (IR) are widely and increasingly being used by researchers and developers. Datasets are essential for such algorithms and the advances in this research area in general. The collection, quality, and availability of such datasets represent an obstacle. This has been reported barrier for researchers in the software engineering domain in general. These challenges motivated us to investigate the possibility of using automated big-data analysis approaches to generate scientific datasets from thousands of open source projects.

- **Goal<sub>4</sub>.** *Classification, and automated categorization and detection of open-source software artifacts*



In recent years, with the advancement and popularity of the open-source approach to software development, researchers benefit from publicly available source code repositories [133]. Software artifacts, other than source code and issue tracking entities, can also provide a great deal of insight into a software system and facilitate knowledge sharing and information reuse. Previous studies show that obtaining such artifacts from open-source projects is non-trivial and researchers lack appropriate automated support to identify, filter, and browse through such artifacts [188]. More importantly, we currently lack an in-depth understanding of the various types of software artifacts that are available in open-source projects. The common assumption is that open-source projects often lack software artifacts such as requirements and design documents.

- **Goal<sub>5</sub>.** *Traceability Datasets Quality Assessment Survey*

Advances in the area of auto-mated software requirements traceability is significantly dependent on the quality of the datasets used to train, test, validate, or tune the underlying machine learning algorithms. Many researchers tried to understand and analyze the importance and impact of datasets quality from different points of view [53, 92, 94, 101, 117, 184]. Yet, few efforts have been taken to understand how experts define quality in traceability datasets and how they assess it.

- **Goal<sub>6</sub>.** *T-DQA Web-Tool* This web-tool is going to be an implementation of our quality framework that will have all quality metrics applied to datasets. In addition, this tool will have filtration parameters that help researchers to easily select and download the datasets that satisfy their research needs.

First, to achieve the first two goals (Goal 1 and Goal 2) in this research project, we performed a systematic literature review (SLR) to assess the current state of software traceability datasets that have been used by researchers in the community over the past fifteen years. Specifically, we investigate 1) the *characteristics* of those datasets, 2) ways to evaluate their *quality*, 3) the *threats to validity* associated with those datasets, 4) factors that are associated with their *reusability*, and 5) the *diversity of datasets* used in the community. In addition, we conduct a large-scale study to identify which types of software artifacts are produced by a wide variety of open-source projects at different levels of granularity.

Through a set of research questions, we aim to explore the diversity, characteristics, and quality of the used datasets. Furthermore, we introduce a Traceability-Dataset Quality Assessment (T-DQA) framework to categorize software traceability datasets and assist researchers to select an appropriate dataset for their research based on different characteristics of the datasets and the context in which those datasets will be used.

Second, to achieve our third goal (Goal 3), we empirically present an empirical study and novel techniques that advance previous work as well as of future software architecture 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 datasets. The Web-Mining technique generates training sets by automatically mining tactic's APIs from technical programming websites. In contrast, the 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 work contains over 116,609 open source projects. Furthermore, we propose an automated approach based on Machine Learning techniques to identify various types of software artifacts. Through a set of experiments, we report and compare the performance of these algorithms when applied to software artifacts.

Third, to achieve this goal (Goal 4), we conduct a large-scale empirical study involving 383 open-source software projects that are randomly sampled from GitHub. These projects are studied to obtain an empirically-based understanding of the artifacts developed in open-source projects. Then we classify all artifacts contained in this sample of open-source projects using the proposed automatic approach. Then, after using heuristics to manually categorize artifacts into two groups, we explore various ML algorithms for software artifact classification. In the last step, we report the performance of our approach to the validation and testing datasets and finally, we classify all artifacts present in the 383 open-source projects and report the prevalence of the different types of artifacts.

Finally, to achieve our final goals (Goal 5, Goal 6), we perform an online survey with 23 practitioners and researchers containing open questions that we analyzed using grounded theory and open coding. In this work, our aim is to understand how software traceability experts evaluate the quality of their datasets. In addition, we aim at gathering experts' opinions on all quality attributes and metrics proposed by T-DQA. Based on the results of this study and the updated quality framework *T-DQA v.2*, We have built a web-tool that utilized these findings and assess the quality of 38 datasets that researchers can choose from based on multiple quality filters to select what matches their needs.

## 1.2 List of Contributions

- Our literature review study highlights the detailed characteristics of the datasets used in the domain of software traceability. This provides an in-depth understanding of the current state of the datasets used in the community and draws their attention to the areas that can improve rigorosity of evaluation and practicality of research.
- In our literature review study, we have presented a new quality assessment framework to reason about traceability datasets, and reveals tacit information about a large number of datasets used in the community which can highlight the path for addressing the threats to validity of the research conducted in this area.
- In our literature review study we also makes the tacit community wide threats related to the datasets explicit. This will help us as a community to better understand the strengths and weaknesses of our empirical foundations, but also, it will help to make more informed decisions in assessing and improving the quality of our datasets.
- T-DQA is the first proposed solution in the community and it relies on the knowledge and frameworks used in other areas of computing. We have collected 73 datasets, evaluated them using the T-DQA framework, and released the results publicly. These results can be used as guidelines for the researchers to select datasets based on their research needs and the characteristics of the datasets.
- In a separate study, we conducted an on-line survey that solicited feedback from 23 software traceability experts. As a result, we proposed *T-DQA v.2* which complements T-DQA based on the feedback collected from the traceability experts. *T-DQA v.2* can be used to characterize existing benchmark traceability datasets and enable the researchers to better reason about the quality of these datasets within the context of their research problems at hand.

- We provided a web-tool that utilizes *T-DQA v.2* to assess the quality of 38 datasets that we have collected along with their traceability artifacts. This web-tool will provide researchers with the datasets that matches their needs while being able to select them based on different quality aspects.
- In our second study, 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.
- We provide a tool called BUDGET (available online<sup>1</sup>) that implements our automated approaches. BUDGET enables traceability researchers to mine a collection of software repositories containing 22 million source files to create training sets. BUDGET also implements several data sampling techniques.
- Our third study of software artifacts detection and classification provide insights into the types of artifacts created during open-source software development. Although documentation related artifacts only account for 6.12% of total software artifacts in open-source software projects, 14.88% of the projects contain either design or requirement documents, which is valuable resources for empirical studies that require such documents.
- We propose a novel approach that utilizes heuristics and various ML classifiers that automatically classify software artifacts.

### 1.3 List of Publications

- W. Zogaan, P. Sharma, M. Mirahkorli, and V. Arnaoudova. Datasets from fifteen years of automated requirements traceability research: Current state, characteristics, and quality. In 2017 IEEE 25th International Requirements Engineering Conference (RE), pages 110–121, Sept 2017.
- Waleed Zogaan, Ibrahim Mujhid, Joanna C. S. Santos, Danielle Gonzalez, and Mehdi Mirakhorli. Automated training-set creation for software architecture traceability problem. Empirical Software Engineering, pages 1–35, 2016.
- J. C. S. Santos, M. Mirakhorli, I. Mujhid, and W. Zogaan. Budget: A tool for supporting software architecture traceability research. In 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA), pages 303–306, April 2016.
- Yuzhan Ma, Sarah Fakhoury, Michael Christensen, Venera Arnaoudova, Waleed Zogaan, and Mehdi Mirakhorli. Automatic classification of software artifacts in open-source applications. In Proceedings of the 15th International Conference on Mining Software Repositories, MSR '18, pages 414–425, New York, NY, USA, 2018. ACM.

---

<sup>1</sup><http://design.se.rit.edu/budget/>

## **1.4 Dissertation Organization**

The remainder of this document is organized as following: Chapter 2 gives a background about software traceability datasets and automation of datasets generation. In Chapter 3, we describes the methodology we followed to address our research questions RQ1, RQ2, RQ3, RQ4, RQ5, RQ6 and RQ7. Chapter 4 presents our work results towards answering our research questions. In Chapter 5, we conclude our work and discuss the contributions.

## Chapter 2

# Background

What is software traceability? “It is simply the potential to relate data that is stored within artifacts of some kind, along with the ability to examine this relationship [44]. It is all about establishing round-trip traceability between different software sources and target artifacts which supports several activities such as architecture-level change-impact analysis, design reasoning, and long-term system maintenance. Different aspects were tackled by researchers to address the challenges in this area. The automation of software traceability is one of the most active research areas where the focus is on utilizing machine learning (ML) and information retrieval (IR) to automate such process instead of the high-cost manual one. These algorithms are data-driven and their accuracy and effectiveness rely mainly on collecting datasets with high quality which leads to other challenges and active research areas in this domain.

In the remaining of this chapter, we discuss five main groups of related work: systematic literature reviews (SLRs) in the domain of software traceability (Section 2.1), the use of Open-Source software as a dataset (Section 2.2), work that assess the quality of datasets (Section 2.3), work on automated datasets generation (Section 2.4), and work on categorization of software artifacts (Section 2.5).

### 2.1 SLRs in Traceability

Several SLRs exist in field of software traceability [23], [135], [158], [45].

Borg et al. [23] conducted an SLR on Information Retrieval-based trace recovery based on 79 publications. However, this study mainly focused on the classification of publications based on the IR techniques used by the authors. In contrast, our study focuses on characterizing the dataset used in the domain of automated software traceability research. Borg et al. briefly discuss that most requirements documents used by researchers had less than 500 requirements, and results were reported only using precision and recall. However, this SLR did not focus on studying the datasets used within the community, therefore, it lacks insights in this regard.

Nair et al. [135] looked at 70 papers related to Software traceability from the International Conference on Requirements Engineering and inspected various aspects of traceability.

The scope of this study was very limited. Regarding the datasets usage, the authors mention that out of 70 papers, 27 (38.7%) do not specify any details about the datasets. They report a rising trend in field traceability with an increasing emphasis on quality of experimentation and academic-industrial partnership.

Santiago et al. [158] conducted an SLR on the impact of Model-Driven Engineering in traceability. Based on 157 studies, they report that storage, data related operations, and visualization are more widely studied aspects of traceability compared to exchange and analysis.

Cleland-Huang et al. [45] analyzed the earlier and current trends in the field of software traceability. They point out some intriguing future research questions concerning the cost-effectiveness, trusted, scalable, portable, ubiquitous, and visualization aspects of traceability techniques. They report that there is a lack of datasets that contain multiple artifact types (e.g., requirements, design, code, test cases, etc.), which in turn leads to limited studies in the direction of automation of traceability link evolution.

In this work, we conduct an SLR that focuses on the characteristics of the traceability dataset as well as their quality. None of the previous studies provide a quality assessment of datasets. We also investigate the reusability of traceability datasets which again is not investigated by the above SLRs.

## 2.2 Open-Source Software as a Dataset

Godfrey and Tu [77] focus on the evolution of open-source software development and examine 96 releases of the Linux operating system kernel. This study aims to compare the evolutionary narratives of open-source with commercially developed systems. However, only files with “.c” and “.h” extensions are examined. Other source artifacts such as configuration files and documentation are ignored.

Behnamghader et al. [20] introduce a framework for conducting large-scale replicable empirical studies of architectural changes across different versions of 23 open-source software systems. The findings of this work bring new insights into the frequency of architectural changes in software systems.

Munaiah et al. [133] propose a framework that helps researchers to identify GitHub repositories that contain engineered software projects. The proposed work defines dimensions that are used to classify software engineered projects by validating the existence of such dimensions in GitHub repositories.

Tian et al. [169] propose a technique using LDA to automatically categorize open-source applications. The proposed technique, called LACT, is evaluated in two studies and the results show that LACT can effectively and automatically categorize software systems regardless of their programming language.

Vendomo et al. [173] conduct an empirical study aiming at identifying and automatically detecting exceptions in open-source software licenses by relying on machine learning. They analyze the source code of 51K projects written in six programming languages and identify 14 different license exception types.

Caniell et al. [31] present a dataset that contains source code and related metadata of FOSS history for the Debian operating system. This dataset contains over 30 million code files in C and C++ along with their related metadata files.

In addition, there are a number of projects in the area of mining open-source software repositories [66, 186] with primarily focus on studying the source code and coding issues. There is a limited experimental research on using such resource to generate scientific datasets with diverse artifacts.

## 2.3 Assessing the Quality of Datasets

Data quality assessment frameworks are adopted in various fields of computing such as requirement engineering [53], information systems [101] [116], web linked data [184], data-warehousing [92] [120], and health-care [139], [94] to assess the quality of scientific datasets. Some of these frameworks are specific to a domain, while others are applicable to broad range of scientific datasets.

Liebchen et al. [106] conducted a systematic review of 23 papers to study accuracy (noisiness) based data quality. They conclude by stating that data quality should be taken into account while selecting a dataset. Another area of focus should be in identifying and correcting noisy datasets along with taking into account the impact of these noisy datasets on the result of the particular studies.

Bosu et al. [24] analyzed papers concerning data quality and identified issues within them. They emphasize the importance of quality in datasets in software engineering case studies. In accordance with [23], [135] and [45], Bosu et al. highlight the need for industrial collaboration to understand and improve data quality.

Wang et al. [174] conducted a market research survey to derive a total data quality framework adhering to the needs of the consumer. They broadly categorized their sub-dimensions and data quality metrics under four labels: Intrinsic, Contextual, Representational, and Accessibility.

Zaveri et al. [184] surveyed 30 existing data quality assessment approaches and derived 18 dimensions and 69 metrics for assessing the quality of linked data. They also draw a comparative analysis of 12 tools based on eight different attributes. They conclude on improving the tools for ease of use and result interpretation.

None of the previous studies considered the quality assessment for software traceability datasets. Our goal is to propose a novel Dataset Quality Framework, we call it Traceability-Dataset Quality Assessment (T-DQA) tailored to the field of traceability, and based on several statistical metrics.



## 2.4 Automated Datasets Generation

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 labeled data points exists and these approaches try to incorporate various sampling [163, 176], instance selection [29, 32, 69] and data reduction techniques [143, 176] to obtain a small representative sample. Unlike these approaches, we do not make such an assumption and the main problem in the area of software traceability is the lack of any labeled data.

In the context of data mining, Zarei et al [183] described the automatic generation of training datasets for classifying Peer-to-Peer (P2P) traffic. The generation of training samples was made through sampling packets from incoming traffic and selecting some of them based on heuristics and statistical models. One advantage of this approach is that it could continuously update the classifier through collecting data periodically, so allowing their classification mechanism to detect new traffic patterns. By applying their dataset to classify incoming packets to a university network, their classifier was able to detect the traffic flow with higher than 98% accuracy when using the generated training data with a false positive rate of about 1.3%. These generated datasets, however, can be applied only for P2P traffic classification while our approach aims to solve the problem of reusable datasets for the software architecture domain.

Other research studies proposed to automatically generate training samples to improve their classification accuracy by increasing the size of the training datasets. For instance, Varga and Bunke [30] showed a training set generation for text recognition of handwritten documents. Their training data was generated through performing geometrical transformations into existing samples of handwritten lines of text. Through a set of experiments, they showed that the use of the generated training data increased the recognition rate of handwritten text. Similarly, Guo and Viktor [80] approached the problem of adjusting unbalanced training data (when the number of samples from one class is significantly higher than the others) through the automatic generation of training samples from existing ones in the form of a short-sized class of datasets. Contrasted to our approach, these solutions still need the manual collection of labeled training samples in order to produce more training data sets whereas our method does not need such samples. In fact, they were concerned about increasing training data size rather than providing a solution to automatically generate training data samples for being reused by classifiers within the same domain.

Several independent software engineering communities are providing mechanisms for publishing and sharing datasets. The Mining Software Repositories (MSR) conference holds a Data Track every year where researchers can publish and share their dataset. The Center of Excellence for Software Traceability holds traceability challenges where researchers can share their datasets related to software traceability challenges. Most works published on these repositories are based on manually created datasets [107]. In our work, we utilize a massive amount of public data on the web and large scale software repositories and provide required automation to create high-quality datasets.

## 2.5 Categorization of Software Artifacts

Robles et al. [153] analyze source code artifacts from versioning repositories beyond source code and provide insights into software projects from both a technical and management point of view. Robles et al. [154] propose a semi-automatic approach that determines the

availability and quantity of documentation and source code comments in a libre software package. In both studies, only file extensions and names are utilized to identify the different types of files. Our approach is complementary to this study since we use file content in addition to file name and extension when classifying artifacts. We use manually extracted features and machine learning algorithms to classify documentation related artifacts thus proposing a fully automated approach.

Gousios and Zaidman [79] introduce pullreqs, a dataset of almost 900 OSS GitHub projects and 350,000 pull requests that are used to study the pull request distributed development model. The main focus of their study is to understand the principles that guide pull-based development. Do et al. [64] design and construct an infrastructure to support controlled experimentation with testing techniques. The infrastructure includes artifacts (programs, versions, test cases, faults, and scripts) that enable researchers to perform controlled experimentation and replications. While these studies provide artifacts that can be used to improve the understanding of one aspect of OSS development, we complement these works by automatically detecting and categorizing multiple OSS artifacts, which can be beneficial to various OSS development activities.

Mirakhorli and Cleland-Huang [129] present an approach using ML to discover architectural tactics in code. The ML classifier is trained using code snippets extracted from OSS systems to automatically detect and categorize code-related files that contain ten common architectural tactics. Our study is not limited to a specific artifact type. Instead, we categorize both documentation related and non-documentation related artifacts, including but not limited to code related files.

Kalliamvakou et al. [95] conduct a study to understand the characteristics of the repositories and users in GitHub. They analyze a GHTorrent dump [73] to identify a set of perils that software engineering researchers should consider when utilizing GitHub repositories in their studies. While this study focuses on the projects and user's characteristics, we analyze and classify software artifacts.

## 2.6 Surveys in software engineering

Several studies solicit experts' opinion in the domain of software engineering and software traceability [13, 14, 25, 115, 178].

Bouillon et al. [25] conducted an online survey to explore the most relevant traceability usage scenarios to practitioners. They identify a list of 29 regularly cited usage scenarios and via an online survey, they asked 56 practitioners to assess how frequently they are using each one in their projects. The results showed that all listed traceability scenarios to be used by practitioners.

Malviya et al. [115] presented an empirical study to elicit and analyze the queries of 29 requirements professionals to understand what questions do they ask when performing requirements engineering tasks. Using an online survey, the data was collected from the participants and analyzed using open coding and grounded theory. The study resulted in identifying 159 questions and 80 different types of artifacts that practitioners use.

To understand why and how often developers rename program identifiers, Arnaoudova et al. [13] conducted a survey with 71 industrial and open-source developers. Results show that developers tend to rename identifiers to improve the quality of the source code lexicon and its consistency with the program functionality. In addition, they proposed an automated approach to document, detect and classify identifier renamings in the source code.

In another work, Arnaoudova et al. conducted two online studies with 44 developers to understand developers' perception of Linguistic Antipatterns, i.e., recurring poor practices related to inconsistencies among the naming, documentation, and implementation of a software entity [14].

Yamashita et al. [178] conducted a survey with 85 professional software developers with the goal of understanding if they consider code smell important. The results showed that 32% of the respondents do not know about code smells.

Many researchers tried to understand and analyze the importance and impact of datasets quality from different points of view [53,92,94,101,117,184]. Yet, few efforts have been taken to understand how experts define quality in traceability datasets and how they assess it. Furthermore, there is a need to introduce metrics and ways of measurements to help researchers better assess the quality of their datasets. Among those, Hayes et al. [86] emphasize the utilization of vetting processes and tools to ensure the dataset's quality. Bosu et al. [24] have developed a taxonomy presenting data quality issues in empirical software engineering. In a previous study [188], we introduced a *Traceability-Datasets Quality Assessment (T-DQA)* framework that defines quality dimensions and metrics for traceability datasets. The framework was defined through the mean of a systematic literature review and adaptation of existing quality assessment frameworks to the domain of software traceability. This is the most relevant work in the domain of software traceability and also the most comprehensive approach. However, to the best of our knowledge, this approach has not been evaluated by experts in the domain of software traceability, nor directly driven from their feedback. Thus it is unclear whether traceability experts agree with the proposed framework and whether it is complete.

These studies all focused on software development activities and requirements engineering tasks. Our focus in this study is to understand what quality attributes experts use to evaluate software traceability datasets and what is their opinion on the existing T-DQA framework. To this end, as the above previous works, we conduct an online survey.

# Chapter 3

## Methodology

In this section, we discuss in detail our research goals along with the main and sub-research questions that we are seeking an answer to achieve each goal. In addition, we describe the methodology to follow in order to achieve our goals and answer our research question.

### 3.1 Research agenda

In order to reach our goals, we plan to go through the following steps:

1. *Investigate the characteristics and types of software traceability datasets that have been used by the researchers within the community:* We aim to assess the existing datasets and have some insight about their diversity and quality. Also, we want to understand the challenges of obtaining such datasets. Therefore, we are seeking an answer to the following research questions:
  - **RQ<sub>1</sub>: What are the statuses, characteristics and types of the existing software traceability datasets that been used by the researchers in the community?**
    - *RQ<sub>1.1</sub>: What are the source and target artifacts in traceability datasets?*
    - *RQ<sub>1.2</sub>: Which application domains are represented by traceability datasets?*
    - *RQ<sub>1.3</sub>: What is the size of traceability datasets?*
    - *RQ<sub>1.4</sub>: What proportion of the traceability datasets is from industry, open-source projects, and student generated data?*
    - *RQ<sub>1.5</sub>: Are traceability datasets available for reuse?*
    - *RQ<sub>1.6</sub>: Is there a relation between the characteristics and the quality of traceability datasets on the one hand and their reusability on the other hand?*
    - *RQ<sub>1.7</sub>: What are the threats to validity associated with traceability datasets?*
    - *RQ<sub>1.8</sub>: Do we, as a community, strive for a diversity of traceability datasets?*

2. *Can we build a data quality framework to help assess the quality of open-source software and industrial projects:* We will investigate existing frameworks used by researchers for evaluating their datasets and how can we adapt those to traceability datasets. In this step, we aim to answer the following research questions:
  - **RQ<sub>2</sub>: How to assess the quality of traceability datasets?**
    - *RQ<sub>2.1</sub>: Is there a relationship between the characteristics and the quality of traceability datasets on the one hand and its reusability on the other hand?*
3. *Investigate the potential of leveraging open source projects and automatically sample and generate useful datasets from them:* Our goal is to investigate the potentials of using automated big-data analysis and web-mining approaches to generate scientific datasets from thousands of open source projects. Also, we aim to compare the quality of the generated datasets with datasets created by experts. First, we will conduct a preliminary study on one traceability scenario. In this step, we aim to answer the following research questions:
  - **RQ<sub>3</sub>: Is it feasible to automatically generate datasets from open-source software repositories?**
    - *RQ<sub>3.1</sub>: Does the training method based on automated web-mining result in higher trace-links classification accuracy compared to an expert-created training set?*
    - *RQ<sub>3.2</sub>: Does the training method based on automated big-data result in higher trace-links classification accuracy compared to an expert-created training set?*
    - *RQ<sub>3.3</sub>: What is the impact of training set size on the accuracy of trace link classification?*
4. *Classification and automated detection and categorization of software artifacts:* The common assumption is that open-source projects often lack software artifacts such as requirements and design documents. In this work, we aim at improving the understanding of open-source projects by investigating this common assumption. Then we propose an automated approach based on Machine Learning techniques to automatically identify various types of software artifacts. In this step, we aim to answer the following research questions:
  - **RQ<sub>4</sub>: Can we automatically detect and categorize open-source software artifacts?**
    - *RQ<sub>4.1</sub>: How can software artifacts be categorized?*
    - *RQ<sub>4.2</sub>: How accurate is the proposed approach for automatic software artifact classification?*
  - **RQ<sub>5</sub>: What types of artifacts are created during open-source software development?**
5. *Traceability datasets quality assessment survey:* In this study, we conduct an online survey with software traceability experts to (i) solicit their opinion about dataset

quality attributes in a data-centric field such as software traceability; evaluate the T-DQA framework (ii) identify the shortcoming of the T-DQA framework through discovering additional measurement and metrics to be added to the framework.

This study sheds light on how experts define quality for traceability datasets and how they assess it. In particular, we seek answers to *what quality aspects they consider the most important when they choose among different datasets*. In this step, we aim to answer the following research questions:

- **RQ<sub>6</sub>**: How do experts assess the quality of traceability datasets?
- **RQ<sub>7</sub>**: Does the existing framework for evaluating the quality of traceability datasets captures the relevant characteristics that experts are looking for?

To achieve the previously mentioned research goals and answer our research questions, we have adopted following methodology:

## 3.2 Goal 1: Investigate the characteristics and types of software traceability datasets:

To go through this step and investigate the characteristics and types of software traceability datasets that been used by the researchers within the community, we conducted a systematic literature review (SLR) that assesses the quality and characteristics of the used datasets in traceability research for the last fifteen years. We performed a preliminary search to retrieve existing literature reviews in the domain of software traceability and potential relevant studies. We found a few SLRs that are discussed in Section 2.1 but none of them address the research questions that we defined. To identify and collect the datasets used in software traceability community, we covered all published full papers with *empirical* and *automated* software traceability *theme*. We followed the guidelines that were established by Kitcheman et al. [97] for SLR in Software Engineering. In this work, by traceability dataset we mean any form of data used by traceability researchers such as training set, testing sets, validation set, answer set, and case studies. This study provided us with knowledge that helped answering our first research question (**RQ1**) where we identified 73 different datasets from 78 different research paper in the traceability domain.

In the followings we present our research questions (Section 3.2.1), the search strategy that we have used to identify relevant publications (Section 3.2.2), the inclusion and exclusion criteria that we used (Section 3.2.3), the overview of the paper selection process (Section 3.2.4), and finally, the details of the data extraction approach from each paper (Section 3.2.5) that allowed us to answer the research questions.

### 3.2.1 Research questions

- **RQ<sub>1</sub>**: What are the statues, characteristics and types of the existing software traceability datasets that been used by the researchers in the community?

To answer this research question we define the following sub-research questions:

- *RQ<sub>1.1</sub>*: *What are the source and target artifacts in traceability datasets?* We will collect the source and target types of the datasets and we will summarize the results by considering all traceability links types as bi-directional.
- *RQ<sub>1.2</sub>*: *Which application domains are represented by traceability datasets?* We will identify the domain of each dataset and we will group the datasets based on their domains.
- *RQ<sub>1.3</sub>*: *What is the size of traceability datasets?* To provide a standardized way of reporting size, we use a metric called *trace space* that provides a proxy for the complexity of a dataset. Trace space,  $D$ , is defined as the product of the size of the source and the size of the target artifacts:

$$D_{TraceSpace} = |D_{Source}| \times |D_{Target}| \quad (3.1)$$

Note that trace space defines the maximum number of trace links between two artifacts.

- *RQ<sub>1.4</sub>*: *What proportion of the traceability datasets is from industry, open-source projects, and student generated data?* We will use frequencies to answer this research question.
- *RQ<sub>1.5</sub>*: *Are traceability datasets available for reuse?* We will investigate whether the datasets are available online.
- *RQ<sub>1.6</sub>*: *Is there a relation between the characteristics and the quality of traceability datasets on the one hand and their reusability on the other hand?* To answer this research question we use Random Forest, a machine learning algorithm used for creating classification and regression trees. Random Forest can be also used for ranking the importance of different features [27]. The process of building a tree is iterative where the goal at each node is to split the data by using one variable only that best differentiates the data with respect to the dependent variable, i.e., create two nodes that are more homogeneous or more pure than the original node. Node purity is calculated using the residual sum of squares. We use the total decrease of node impurity, *IncNodePurity*, as an indication of the variable importance. A higher value of *IncNodePurity* indicates more important input variable.
- *RQ<sub>1.7</sub>*: *What are the threats to validity associated with traceability datasets?* We will categorize and summarize the threats to validity related to the usage of datasets, acknowledged or mitigated by the studied papers.
- *RQ<sub>1.8</sub>*: *Do we, as a community, strive for a diversity of traceability datasets?* This would provide an insight on whether the same datasets are used for all the research problems in hand or whether we seek to adopt new datasets for different research problems. For authors that have published more than one papers, we calculate a ratio that represents the diversity of the datasets that they use. The diversity ratio for author  $i$ , *DiversityRatio<sub>i</sub>*, is defined as the number of unique datasets, *UniqueDataset<sub>i</sub>*, divided by the total number of datasets, *TotalDatasets<sub>i</sub>*, used across the publications of author  $i$ .

$$DiversityRatio_i = \frac{UniqueDataset_i}{TotalDatasets_i} \quad (3.2)$$

### 3.2.2 Search strategy

A search strategy is fundamental for any SLR to ensure that all relevant studies are considered for accurate conclusions [97, 185]. Our search strategy consists of the following main elements: *search methods*, *search terms*, and *data sources*. We performed a preliminary search to retrieve existing literature reviews in the domain of software traceability. We found a few SLRs that we discuss in Section 2.1 but none of them address the research questions that we defined.

Table 3.1: Venues used in the manual search phase.

<b>Conferences</b>	
ICSE	International Conference on Software Engineering
TEFSE	International Workshop on Traceability in Emerging Forms of Software Engineering
ASE	International Conference on Automated Software Engineering
ESEC	European Software Engineering Conference
FSE	International Symposium on Foundations of Software Engineering
SST	International Symposium on Software and Systems Traceability
RE	International Requirements Engineering Conference
REFSQ	International Working Conference on Requirements Engineering: Foundation for Software Quality
COMP-SAC	IEEE Computer Society International Conference on Computers, Software and Applications
ICSM	International Conference on Software Maintenance
MSR	International Conference on Mining Software Repositories
WICSA	Working IEEE/IFIP Conference on Software Architecture
ICPC	International Conference on Program Comprehension
ECSA	European Conference on Software Architectures
<b>Journals</b>	
EMSE	Empirical Software Engineering Journal
TSE	IEEE Trans. on Software Engineering Journal
ISSE	Innovations in Systems and Software Engineering Journal
SEP	Journal of Software: Evolution and Process
TOSEM	ACM Trans. on Software Engineering and Methodologies
REJ	Requirements Engineering Journal

We conducted our search using both *manual* and *automatic* methods to ensure that we cover as many relevant venues and electronic data sources as possible [185]. In the *manual search*, we went through all papers published in the venues listed in Table 3.1. We built an initial list of relevant venues that we augmented by contacting traceability experts for suggestions on other related sources (conferences, journals, research groups active in this domain or individual papers). Table 3.1 contains venues that are considered high quality venues for software requirements (e.g., RE, TEFSE, and REJ), while other venues have a broader and generic theme (e.g., ICSE, TSE, and TOSEM). In the *automatic search*, we defined a set of search terms. We started with key terms used in traceability papers such as *requirements* and *traceability*. To be as generic as possible we expanded the search terms to include *software traceability* which resulted in our final query as follows: *(software OR requirement) AND traceability*. We used the search terms during the automatic search to search in the electronic data sources listed in Table 3.2 by matching the terms with the title, keywords, and abstract of each paper.



Table 3.2: Databases used in the automatic search phase.

ID	Database	Web address
1	ACM DL	http://portal.acm.org
2	IEEE Explorer	http://www.ieee.org/web/publications/ xplora
3	SpringerLink	http://www.springerlink.com
4	ScienceDirect	http://www.elsevier.com

Table 3.3: Inclusion and exclusion criteria.

Inclusion criteria	
I1	A Full paper.
I2	Focus on software (requirements) traceability.
I3	Proposed/used/evaluated an automated traceability technique.
I4	Used data-sets in their study.
Exclusion criteria	
E1	Position papers, short papers, tool demo papers, keynotes, reviews, tutorial summaries, and panel discussions.
E2	A study that is not written in English.
E3	Duplicated studies.
E4	No datasets or case studies.

### 3.2.3 Inclusion and exclusion criteria

The inclusion and exclusion criteria are specified in Table 3.3 and were applied at different stages to all of the retrieved studies (See Figure 3.1). To limit the scope of our SLR, we included all studies that were published between 2000 and 2016. We have selected all studies that have used datasets, case studies or empirical data to develop, validate, train, or test traceability techniques. Papers that only presented approaches or an idea without empirical data-based validation were excluded. We considered only studies that focus on automated software traceability, therefore, papers related to models and processes of software traceability were excluded. In addition, we excluded short papers, workshops, and tool demonstration papers. Lastly, all duplicated studies found from different sources were identified and removed.

### 3.2.4 Study selection process

Figure 3.1 shows the number of studies selected at each stage of the SLR. The initial search process resulted in 1011 and 5398 papers that were collected during the manual and automatic search, respectively. Because of a large number of retrieved papers (6409), we selected the first set of papers that could be relevant to our study by reading their title, keywords, and abstract [28]. From 6409 papers, we selected 202 papers as the primary studies. All these 202 papers were reviewed by two of the authors of this paper. In this review process, the inclusion/exclusion criteria were applied and the rationale for these decisions was documented. The rationale for including/excluding the studies were reassessed and discussed in separate group discussions. If a paper satisfied all inclusion criteria, it was considered as a *primary study* and was included in SLR. Our final list included 78 papers.

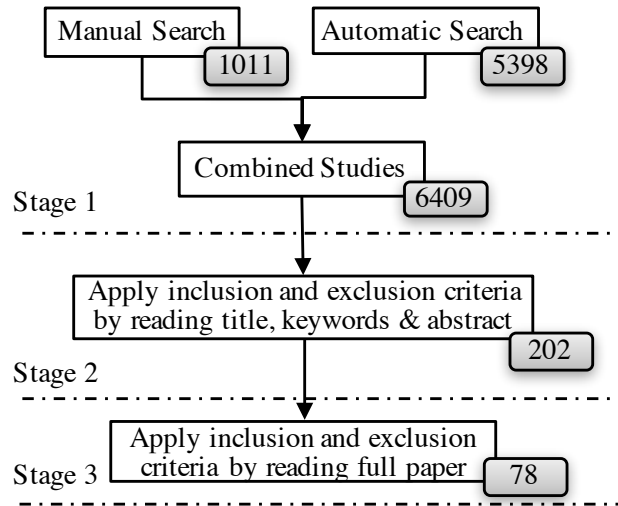


Figure 3.1: Search process stages.

Table 3.4: Extracted dataset items.

Data item	Related RQs
The <b>Name &amp; Traceability Artifacts</b>	RQ1, RQ1.1
The <b>Application Domain</b>	RQ1, RQ1.2
The <b>Size (trace space)</b>	RQ1.3, RQ2
The <b>Type</b> (private, student, or open source)	RQ1.4, RQ2
The <b>Availability</b> and link to the source	RQ1.5, RQ2
The <b>Licensing, Storage, Developer, Programming Language</b>	RQ2, RQ3
The <b>Threats to validity</b> related to the datasets that are acknowledged or mitigated in the paper.	RQ4

### 3.2.5 Data extraction

In the data extraction phase, we collected all information from the selected studies that was necessary to answer our research questions. First, we extracted basic information about the papers such as the list of the author(s), year of publication, title, venue, and publication type (full paper, short paper, etc.). Then, through a set of group studies, we identified, extracted, and organized the information about the datasets used in each research paper. Table 3.4 provides an overview of the dataset items that were extracted from each paper and the research questions that require those items.

## 3.3 Goal 2: Building data quality framework:

For our second step, "building a data quality framework", to answer our second research question (**RQ2, RQ2.1**) we are looking into the data extracted from our SLR study and what metrics been used by the researchers to assess their datasets quality. Also, we are looking into other domain quality assessments metrics that could potentially be used in the traceability domain to assess software traceability datasets.

We have studied several data quality frameworks and looking into adapting them to the domain of software traceability so that we can assess the quality of traceability datasets. As a result, we can propose a Traceability-Data Quality Metric framework that is suited for assessing datasets in the field of traceability.

## 3.4 Goal 3: Feasibility of automatically generate datasets from open source software repositories:

For our third step and to answer its research questions (**RQ3.1, RQ3.2, RQ3.3**), we conducted a preliminary study with limited scope and covered only architecture tactics datasets [187]. In this work, we presented three baseline approaches for the creation of training data for the problem of tracing architectural concerns. 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. Development of such approaches relies on the existence of large, (un)structured and rich knowledge bases. Since both The Web and ultra-large-scale code repositories have such characteristics, one key novelty of the proposed work in this study is to utilize such resources and develop new techniques to help scientists in the area of software architecture traceability to obtain high-quality datasets. The manual dataset that we used was established by experts in previous work by M. Mirakhorli et al. [130,131]. They used a manual approach to collect datasets to train their tactic classifier. The training set 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 dataset of files implementing architectural tactics was discovered through rigorous search and validation approach that resulted in a high quality and precise traceability dataset. However, the cost associated with this approach is substantially high. For instance, it took them about 3 months to collect and peer review tactical data from 10 different projects for 5 architectural tactics. In our study, we compared the trace-link creation accuracy achieved using each of the three baseline approaches and discuss the costs and benefits associated with them.

### 3.4.1 Research questions

- **RQ<sub>3</sub>: Is it feasible to automatically generate datasets from open source software repositories?**

To answer this research question we define the following sub-research questions:

- *RQ<sub>3.1</sub>: Does the training method based on automated web-mining result in higher trace-links classification accuracy compared to an expert-created training set?*

Through a set of experiments, we investigated this research question. Our empirical analysis indicates that the web-mining approach presented in this work 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.

- *RQ<sub>3.2</sub>: 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. In two experiments out of five, the accuracy of trace-link classifier trained using the Big-Data Analysis approach was better than the classifier trained using expert-created dataset. Overall, 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 files. Manually creating such dataset is very time consuming and our automated technique provides a significant reduction in training set creation time

- *RQ<sub>3.3</sub>: What is the Impact of Training Set Size on the Accuracy of Trace Link Classification?*

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.

### 3.4.2 Study scope

As we mentioned earlier that this is a preliminary study with limited scope and covered only architecture tactics datasets. The goal was to investigate the feasibility of using automated techniques to generate useful training-sets with similar quality to manual-created datasets.

Tactics serve as a building block of software architecture and are used to satisfy a specific quality. A definition of tactics is provided by Bachman et al. [15] 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 [16]:

- **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.

### 3.4.3 Traceability challenge: identifying Tactic-Related classes

In previous work [126,131] M. Mirakhorli et al presented a novel approach for tracing architectural tactics to the source code. As a tactic can be implemented in several ways, through structuring the source code in many different forms, we cannot use structural analysis as the primary means of identification. Their approach therefore relied primarily on information-retrieval (IR) and machine learning techniques to train a classifier to recognize tactics in the source code through learning the specific terms that occur commonly across implemented tactics. A tactic-classifier was used to identify all classes related to a given tactic, thereby establishing traceability links from tactics to the code [128]. The classifier needs to be first trained by several sample implementation of each tactic, and it includes three phases of preparation, training, and classification which are defined as follows:

**Data preparation phase:** In this phase, the source code of both the training set and the system under test are preprocessed using standard information retrieval techniques (stemming, stop terms removal, etc). This way, each source code is represented as a vector of terms.

**Training phase:** The training phase takes a set of pre-classified code segments (i.e., training set) as input, and produces a set of *indicator terms*, that are considered to be representative of each tactic type, along with a *weight score*, which shows the level of importance of a specific indicator term with respect to the tactic. 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. In short, the weight score is the probability that a given term is related to a specific tactic in the training set provided as input.

The training formula has three parts. The first part normalizes the frequency with which term  $t$  occurs in the training document with respect to its length. The second part computes the percentage of training documents of type  $q$  containing term  $t$ . Lastly, the third part decreases the weight of terms that are project-specific.

More formally, let  $q$  be a specific tactic such as *heartbeat*. 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  $N_q$ . In equation 3.3  $N_q(t)$  refers to the total number of tactic related training files containing the term  $t$ , while  $N(t)$  refers to the total number of documents containing term  $t$ .  $NP_q(t)$  refers to the total number of tactical projects containing term  $t$  and  $NP_q$  refers to the total number of projects in the training data. Through the training process, 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:

$$Pr_q(t) = \frac{1}{N_q} \sum_{c_q \in S_q} \frac{freq(c_q, t)}{|c_q|} * \frac{N_q(t)}{N(t)} * \frac{NP_q(t)}{NP_q} \quad (3.3)$$

After calculating the weight score ( $Pr_q(t)$ ) for each term  $t$ , this phase consider that a term is an *indicator term* if it is higher than a fixed value (*term threshold*).

**Classification phase:** During the classification phase, the indicator terms computed in Equation 3.3 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:

$$Pr_q(c) = \frac{\sum_{t \in c \cap I_q} Pr_q(t)}{\sum_{t \in I_q} Pr_q(t)} \quad (3.4)$$

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  $q$ . This score  $Pr_q(c)$  indicates how similar the code is to an implementation of the tactic. Hence, if this probability is higher than a fixed value (*classification threshold*), the code is considered as tactical file.

### 3.4.4 Overview of the three baseline techniques

As previously stated, the scope of this project is to 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.

In a series of experiments we rigorously evaluate and compare three baseline training set creation techniques: (i) Manual Expert-Created approach, (ii) Automated Web-Mining approach which creates training-set through mining technical API libraries on the World Wide Web, and (iii) Automated Big-Data Analysis technique which creates code based training-set through mining code snippets from ultra-large-scale source code repositories.

The proposed automated training set creation techniques as well as the traditional expert-created approach are illustrated in Figure 3.2. In the case of the manual expert-created approach, architects collect, review and refine the training set. In the case of the automated techniques, a description of the tactic from textbooks (or a set of tactic related terms) can be used as a tactic-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 sections we describe each of these baseline techniques. Then in section 3.4.8 we report empirical studies conducted to compare them.

### 3.4.5 Baseline method 1: expert-created approach

In previous work [130,131] M. Mirakhorli et al. used a manual approach to collect datasets to train our tactic classifier. The training set shown in Table 3.5 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 files 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 tactic-query for each tactic was composed of keywords used in

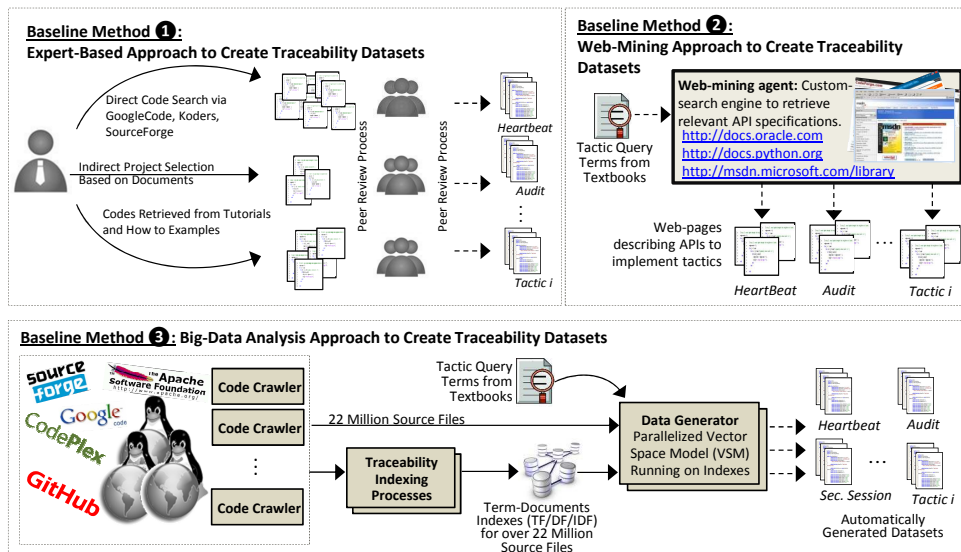


Figure 3.2: Overview of Automated Approaches to Create Tactic Traceability Training-sets

Table 3.5: Manual Dataset Generated by Experts

<b>Tactic</b>	<b>Projects</b>
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-Realtime 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 Network-Manager(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)



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 files 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 files 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. For each of the project, they identified (i) if the tactic is implemented in the project, (ii) which files are involved in the implementation of the tactic, and (iii) how and why the tactic is being used in the project (rationale for the design decision). Through this process we eliminated cases that the tactic was used outside its intended context. This dataset is released at <http://coest.org/mt/27/150>.

### 3.4.6 Web-mining approach

Web based libraries, such as *msdn*<sup>1</sup> or *oracle*<sup>2</sup>, 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 3.3(a) and 3.3(b) illustrates sample implementation guidelines retrieved from these libraries to implement reliability requirements through *Heartbeat* and security requirements through *Audit Trail* tactics.

#### Data collection agent

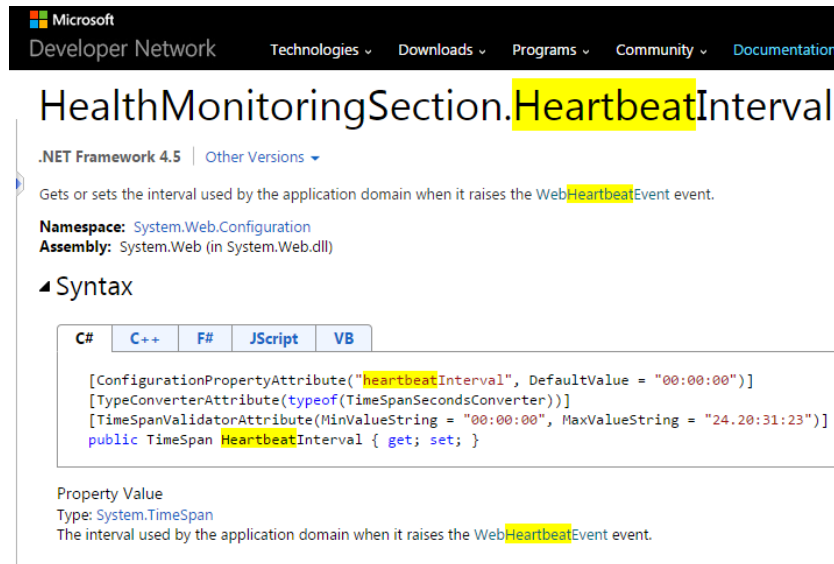
We developed a custom web scraper which uses the search engine APIs of Google to query the content of predefined technical libraries (e.g. *msdn* and *oracle*).

The tactic-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 [16]: “**Heartbeat** is a **fault detection** mechanism that employs a periodic message exchange between a system **monitor** and a process being monitored.” We generated the following *trace query* from this description: *Heartbeat* OR *fault* OR *detection* OR *monitoring*. Although the tactic queries

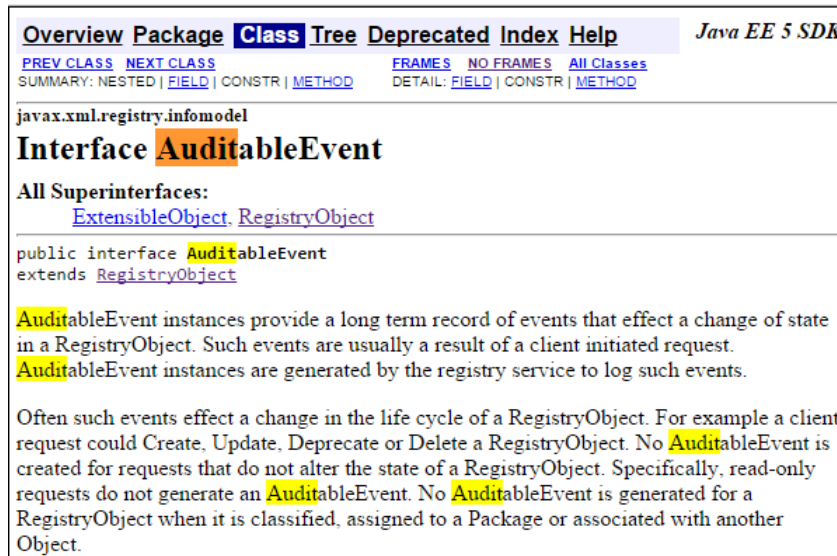
---

<sup>1</sup><https://msdn.microsoft.com>

<sup>2</sup><http://www.oracle.com>



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



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

Figure 3.3: Two sample API descriptions from technical libraries of (a) MSDN and (b) Oracle

can be more complex than OR joints of search terms, in this work we only use simple tactic-queries.

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.

### Generated data

The generated data to train the classifier is a balanced dataset, containing the same number of positive and negative samples. In this case, a balanced samples of 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 files. 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).

### 3.4.7 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 3.2.

#### 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 GHTorrent<sup>3</sup> 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* [171], 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 (i.e. projects

---

<sup>3</sup><http://ghtorrent.org/>

that have less than 20 source files). Table 3.6 shows the frequency of all the projects in different languages in our repository.

Table 3.6: Overview of the projects in Source Code Repository of Big-Data Analysis Approach

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

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

### 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 [123].

### Data generator component

The third component is a paralleled version of Vector Space Model (VSM) [156] capable running over 22 million source files in a few seconds. The VSM is a standard approach in which a query  $q$  and a source file  $f$  are both represented as a vector of weighted terms. Therefore, a source file  $f$  is represented as a vector  $\vec{f} = (w_{1,f}, w_{2,f}, \dots, w_{n,f})$  and a query  $q$  is represented as  $\vec{q} = (w_{1,q}, w_{2,q}, \dots, w_{n,q})$ , where  $w_{i,f}$  represents the weight of the term  $i$  for source file  $f$ . We used the standard weighting scheme known as  $tf - idf$  to assign weights to individual terms [156]. In this scheme, the  $tf$  represents the term frequency, and the  $idf$  corresponds to the inverse document frequency. The term frequency is computed for source file  $f$  as  $tf(t_i, f) = (freq(t_i, f)) / (|f|)$ , where  $freq(t_i, f)$  is the frequency of the term in the file, and  $|f|$  is the length of the file. The inverse document frequency  $idf$ , is typically computed as :

$$idf_{t_i} = \log_2 \frac{n}{n_i} \quad (3.5)$$

where  $n$  is the total number of source files in the corpus (our repository) and  $n_i$  is the number of source files in which term  $t_i$  occurs. Thus, the individual term weight for term  $i$  in source file  $f$  is then computed as  $w_{id} = tf(t_i, f) \times idf_{t_i}$ . Given these definitions, the similarity score  $Sim(f, q)$  between a source file  $f$  and technical query  $q$  is computed as the cosine of the angle between the two vectors as:

$$Sim(f, q) = \frac{(\sum_{i=1}^n w_{i,f} w_{i,q})}{\left( \sqrt{\sum_{i=1}^n w_{i,f}^2} \cdot \sqrt{\sum_{i=1}^n w_{i,q}^2} \right)} \quad (3.6)$$

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, using the formula in Equation 3.6, 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 work [47, 126, 131].

### Generated data

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

#### 3.4.8 Experiment overview

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.

### Justification for selection of approaches

The domain of automatically-generated training sets for machine learning 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.

The development of such approaches relies mainly 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 our proposed work is to utilize such resources and develop new techniques to help scientists in the area of software architecture traceability to obtain high quality datasets.

### Expert-created 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 ((random sampling of tactical files)) 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 files and 10 non-tactical ones.

## 3.4.9 Experiment design

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

### Experiment design for using baseline method 1

The accuracy of classification techniques trained using the expert-created approach was evaluated using a standard 10-fold cross-validation process. In this experiment the expert-created 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 [41, 43, 98, 131].

In each execution, the data was partitioned by project such that in the first run nine projects, each including one related and four unrelated files, 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 execution.

### 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 expert-created dataset of files established by experts (Table 3.5). The experiment was repeated using a variety of term thresholds and classification thresholds required for equations 3.3 and 3.4. The term thresholds were used

for deciding which terms should be part indicator terms list and the classification thresholds were utilized to classify a given source file into tactical/non-tactical.

### 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 expert-created dataset of tactical files collected by the experts. The training data was sampled from over 116,609 open source projects in our code repository.

#### 3.4.10 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 *tactical code files*.

$$Recall = \frac{|RelevantCode \cap RetrievedCode|}{|RelevantCode|} \quad (3.7)$$

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

$$Precision = \frac{|RelevantCode \cap RetrievedCode|}{|RetrievedCode|} \quad (3.8)$$

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:

$$FMeasure = \frac{2 * Precision * Recall}{Precision + Recall} \quad (3.9)$$

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

$$Specificity = \frac{|NonRelevantCode|}{|TrueNegatives| + |FalsePositives|} \quad (3.10)$$

#### 3.4.11 Minimizing biases

To avoid the impact of datasets size, all the datasets that were automatically generated from our automated approaches (Big-data and Web-mining) included 10 projects, (or 10 related web-pages). We trained the classifier using the files automatically extracted using our own primitive big-data analysis technique and then attempted to classify the expert-created dataset of manually established and reviewed files.

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 4.3.11.

### 3.5 Goal 4: Classification, automated categorization, and detection of open-source software artifacts:

In this step and to answer its research questions (**RQ4.1**, **RQ4.2**, **RQ5**), we conducted a large-scale empirical study using open-source software projects that are randomly sampled from GitHub. The goal behind studying these projects is to obtain an empirically-based understanding of the artifacts developed in open-source projects. Then we classify all artifacts contained in this sample of open-source projects using the proposed automatic approach.

#### 3.5.1 Research questions

- **RQ<sub>4</sub>**: Can we automatically detect and categorize open-source software artifacts?

To answer this research question we define the following sub-researchquestions:

- *RQ<sub>4.1</sub>*: *How can software artifacts be categorized?* To answer this question we randomly sample from a large set of open-source projects and manually examine the type of artifacts available. During this process, we iteratively identify heuristics and features that can be used to automatically classify artifacts.
- *RQ<sub>4.2</sub>*: *How accurate is the proposed approach for automatic software artifact classification?* We investigate the performance of the proposed approach using different evaluation metrics. We report results on validation and testing datasets using 10-fold cross-validation.

- **RQ<sub>5</sub>**: **What types of artifacts are created during open-source software development?** We classify all artifacts present in the studied open-source projects and report the prevalence of the different types of artifacts.

#### 3.5.2 Study definition and design

The *goal* of this work is to investigate what types of artifacts are created during open-source software development. To achieve this goal, we propose an automatic approach for software artifact detection and classification using machine learning approaches. The *quality focus* is the performance of the proposed approach on artifact classification in terms of selected evaluation metrics such as precision and recall. The *perspective* of the study is that of researchers, who are interested in automatically obtaining software development artifacts that fit their research need. The evaluation is carried out in the *context* of open-source projects collected from GitHub [90].

Figure 3.4 depicts the overview of our approach, which is designed to automatically classify software artifacts leveraging (i) heuristics based on file names and extensions and (ii) existing ML algorithms. To answer **RQ<sub>4.1</sub>**, we collect a large set of diverse open-source projects and obtain a significant random sample of the projects. We identify the artifacts contained in the sampled projects and divide them into two groups by applying heuristics on file names and extensions. The first group contains artifacts that can be classified solely based on file names and extensions whereas the second group contains artifacts that require deeper analysis in order to be classified. We manually classify a sample of the artifacts



contained in the second group to construct an oracle of classified artifacts. During the manual classification, we also identify features that could be used to automate the artifact classification. For **RQ<sub>4.2</sub>**, we automate the feature extraction process and use various ML algorithms to automatically classify software artifacts belonging to the second group. Finally, to answer **RQ<sub>5</sub>** we classify all artifacts of the studied open-source projects and report the frequency of occurrence of each type of artifact identified during the manual process.

### 3.5.3 Subject Systems

We extract a large set of 91,108 open-source projects from GitHub making use of a code crawling application known as *GHTorrent* [78]. *GHTorrent* acts as a service to extract data and events, returning 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.

To collect a significant sample of projects for our study, we randomly sample 383 projects from the collected open-source projects, ensuring 95% confidence level and 5% margin of error. All research questions are addressed using the sampled projects.

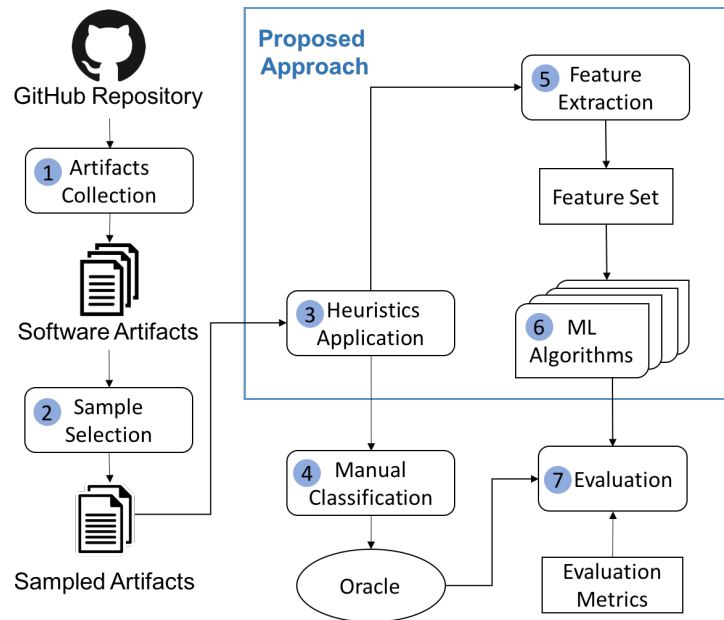


Figure 3.4: Approach overview.

### 3.5.4 Oracle

To create an oracle of classified software artifacts, we manually examine a random set of artifacts from the 383 sampled projects. When the file name/extension are insufficient to classify an artifact, we analyze the file content. Two coders perform the classification of artifacts independently. An inter-rater reliability (IRR) analysis [84] is used to assess the degree to which coders consistently classify software artifacts. Both coders are Master students in Computer Science. Disagreements between the coders are resolved with discussions and when necessary a third coder is brought in. The category of artifacts are coded using categorical variables. The Cohen's kappa statistic measures the observed level of the agreement between coders for a set of nominal ratings and corrects for agreement that would be expected by chance, providing a standardized index of IRR that can be generalized across studies [84]. Possible values for kappa range from -1 to 1, with 1 indicating a perfect agreement, 0 indicating a completely random agreement, and -1 indicating a total disagreement. Landis and Koch [103] provide guidelines for interpreting kappa values as follows: values from 0.0 to 0.2 indicate slight agreement, values from 0.21 to 0.40 indicate fair agreement, 0.41 to 0.60 indicate moderate agreement, 0.61 to 0.80 indicate substantial agreement, and 0.81 to 1.0 indicate almost perfect or perfect agreement. The data in this study is collected through ratings provided by coders and has a significant impact on the computation and interpretation of our study. It is important that coders can independently reach similar conclusions about the types of software artifacts they identify because that confirms the established categories are well defined. Thus, we target at least substantial agreement, i.e., above 0.61.

### 3.5.5 Automatic Artifact Classification

To automate the software artifact classification process we identify heuristics based on file names and extensions (Section 3.5.5). For files that require further analysis we extract features (Section 3.5.5) that we use as input to machine learning algorithms (Section 3.5.5).

#### Heuristics Application

We utilize existing file name/extension categorization [52] and we randomly sample a portion of the most frequently occurring extensions to confirm the correctness of such categorization. In addition to file extension, we expect the file name to provide useful information in artifacts identification as well. For example, testing code is often organized under directory with names contain "test" or "tests" and files with .wav extension can be automatically identified as audio file. Such identification is assumed to be correct by construction. On the other hand, some files, such as .txt, can not be identified without examining the file content.

#### Feature Creation Process

Generating a set of features for text classification problems could be achieved with the use of various information retrieval techniques. For instance, one could use a Vector Space Model [157] and use a weighting schema such as Term Frequency-Inverse Document Frequency (TF-IDF) [164] to automatically extract the most important terms in a document. Other, more sophisticated techniques that could be used are Latent Semantic Indexing (LSI) [65] and Latent Dirichlet Allocation (LDA) [21]. Information retrieval techniques are most useful when the characteristics of the documents that we are working on are unknown.

In other words, we rely on the technique to identify hidden patterns that characterize each document.

Instead, we decided to use the knowledge gained through the manual validation process of artifacts and thus manually creating the set of features that characterize each type of artifact. Because an optimal set of features cannot be determined a priori, the two annotators generate an initial set of features and iteratively refine the set through discussions. This manual approach gives us more flexibility in determining the relevant set of features, while harnessing the knowledge gained during the oracle creation process.

### Machine Learning Algorithms

We select seven different machine learning approaches belonging to three different categories: decision trees, Support Vector Machines, and Bayesian Networks. Research has shown that these algorithms perform well for text classification problems [93, 118, 122, 170]. We use the implementations provided through Weka [83] and evaluate the classifiers using 10-fold cross-validation. In other words, we evaluate the predictive models by partitioning the original sample into 10 equal sized sub samples, performing the analysis on one subset, and validating the analysis on the other. The validation is repeated 10 times to obtain an average estimate of the predictive model. We briefly describe the selected algorithms and the parameter tuning that we performed:

1. **Random Forest** [27] averages the predictions of a number of tree predictors where each tree is fully grown and is based on independently sampled values. The large number of trees avoids over fitting. Random Forest is known to be robust to noise and to correlated variables. We use the function `randomForest` (package `randomForest`) with the number of trees being 500 as a starting point, which has shown good results in previous works [175]. We tune the parameters for the number of trees varying from 500 to 1000 and for the features explored at each branch from the default value:  $(\log_2(\#predictors) + 1)$  to 20% of the total number of features with a step of 0.05.
2. **Sequential Minimal Optimization (SMO)** is an implementation of John Platt's sequential minimal optimization algorithm to train a support vector classifier. We use RBF kernel, Polynomial kernel, and the Pearson VII function-based universal kernel (PUK) [172] in combination with this classifier. We tune the exponent parameter of the classifier varying from 1.0 to 4.0 with a step of 0.5, the gamma parameter from 0 to 1 with a step of 0.05, and the cost parameter from 1 to 50 with a step of 1.
3. **Multinomial Naïve Bayes** is a specific version of Naïve Bayes, created for improved performance on text classification problems [122]. Naïve Bayes is the simplest probabilistic classifier applying Bayes' theorem. It makes strong assumptions on the input: the features are considered conditionally independent among each other. We explore the performance of the classifier using kernel estimator and supervised discretization.
4. **J48** is an implementation of the C4.5 decision tree. This algorithm produces human understandable rules for the classification of new instances. The implementation provided through Weka offers three different approaches to compute the decision trees, based on the type of the pruning techniques: pruned, unpruned, and reduced error pruning. We tune the parameter for the minimum number of instances at each leaf from 1 to 8 with a step of 1.

5. **Ensemble Learning** is used to combine individual classifiers with the aim of obtaining better overall predictive performance. We use the majority vote algorithm provided through Weka. The majority vote approach considers the votes of each classifier for the label of an instance and uses the label agreed upon by the majority.

### 3.5.6 Evaluation

We evaluate the performance of the automatic artifact classification approach using the following evaluation metrics:

#### Precision

Precision is defined as the percentage of artifact predicted as belonging to the categories that are correct with respect to the oracle,  $Precision = TP/(TP + FP)$ , where  $TP$  and  $FP$  are the number of true and false positives, respectively.

#### True Positive Rate (TPR)

TPR or relative recall is calculated as the ratio between the number of true positives and the total number of positive events, i.e.,  $TPR = TP/(TP + FN)$ . In the context of this study, the TPR indicates how many of the manually known software artifacts are correctly discovered.

#### F-Score

Precision and recall are inversely related, thus, it is difficult to compare results of the model using the two metrics. F-score is used to aggregate both measures into a single value. F-score is the harmonic mean of the precision and recall, i.e.,  $F = 2 * Precision * TPR / (Precision + TPR)$ . F-score reaches its best value at 1 (perfect precision and recall) and worst at 0.

#### Area Under the Receiver Operating Characteristic (ROC) curve

ROC is a plot of the true positive rate against the false positive rate at various discrimination thresholds. The area under ROC is close to 1 when the classifier performs better and close to 0.5 when the classification model is poor and behaves like a random classifier.

#### Matthews Correlation Coefficient (MCC)

MCC is a measure used in machine learning to assess the quality of a two-class classifier especially when the classes are unbalanced [121].

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(FN + TN)(FP + TN)(TP + FN)}}$$

Values range from -1 to 1, where 0 indicates that the approach performs like a random classifier. Other correlation values are interpreted as follows:  $MCC < 0.2$ : low,  $0.2 \leq MCC < 0.4$ : fair,  $0.4 \leq MCC < 0.6$ : moderate,  $0.6 \leq MCC < 0.8$ : strong, and  $MCC \geq 0.8$ : very strong [48].

### Micro and Macro Average

There are different ways to average results of a multi-class classifier. Macro-average treats each class with equal weight and is calculated as the average of the metrics computed within each class. Micro-average gives each individual instance equal weight so that the largest classes have most influence. It is computed by aggregating the outcomes across all classes and computing a metric with aggregated outcomes. We report all evaluation metrics along with both micro and macro average.

## 3.6 Goal 5: Traceability datasets quality assessment survey:

For our last step and to answer its research questions (**RQ6**, **RQ7**), To achieve this goal, we conducted an on-line survey that solicited feedback from 23 software traceability experts. The responses were analyzed systematically using grounded theory approach. The *goal* of this study is to gain insight into how experts in the domain of software traceability evaluate the quality of their datasets. The *quality focus* is the importance and completeness of characteristics captured by the Traceability-Datasets Quality Assessment (T-DQA) framework. The *perspective* of the study is that of researchers and practitioners who are interested in mitigating potential threats to validity related to the quality aspect of the datasets that they are using.

### 3.6.1 Research questions

- **RQ<sub>6</sub>**: How do experts assess the quality of traceability datasets?

To answer this research question we define the following sub research questions:

- RQ6.1: *What are the quality attributes that researchers are looking for when they select datasets?*
- RQ6.2: *What dataset qualities have an impact on the meaningful conclusions being drawn from a research project?*
- RQ6.3: *What are the datasets quality-attributes that could impact the generalizability of research results?*

- **RQ<sub>7</sub>**: Does the existing framework for evaluating the quality of traceability datasets captures the relevant characteristics that experts are looking for?

### 3.6.2 Survey Design

Interviews and surveys are two popular data collection techniques in empirical research [145, 150, 162]. We decided to conduct an on-line survey as it minimizes the effort of collecting data, increases the number of potential participants in the study, and, unlike interviews, it allows the participants to answer the questions over a period of time based on their own availability, preference, and schedule [145].

When designing the survey, we focused on using a clear and understandable language within the questionnaire. In addition, to minimize the time of the survey, we used a mix

of multiple choice and text-entry questions. The online questionnaire was implemented in English using a Qualtrics Survey. The survey link was shared through email.

The survey has three main parts. The first part of the survey focuses on gathering demographic information about participants. The second part of the survey contains a series of open-questions aiming to answer **RQ<sub>6</sub>**. To answer **RQ<sub>7</sub>**, the third part of the survey is designed with a series of open and multiple choice questions asking participants to provide feedback on the importance of each characteristic of the T-DQA framework. The multiple choice questions use a 5-point Likert scale [138] ranging from 'Very Important' to 'Not Important'. For each question, participants can decide not to answer or to choose the option 'Do not have an opinion'.

### 3.6.3 Participants

In this survey study, we targeted traceability experts, i.e., researchers and practitioners from either academia or industry with traceability research that involve datasets. We created a mailing list of authors who have published full papers in the area of software traceability. To do so, we utilize the papers included in three recent systematic literature review of traceability papers [23, 135, 188] published from 2000 to 2016.

This process results in a list of 176 potential candidates. From all candidates contacted for the survey, 40 visited the survey link and started the process of answering the survey questions. From these 40 candidates, 23 subjects completed the survey and answered all questions.

### 3.6.4 Pilot Study

To test the clarity of the questionnaire and estimate the time to answer all questions, performing a pilot study is recommended [180]. Therefore, we performed a pilot study with a couple of colleagues who are familiar with software traceability. The feedback we received helped us in correcting linguistic and structural issues in the survey and reduce the number of questions.

### 3.6.5 Data Collection

In order to ensure high data quality, we decided to exclude partly answered questionnaires. We think that participants who answered only the first few questions will not spend sufficient time to think about the questions and thus will not be able to provide real insights into the topic. Thus, we only analyze the data from the 23 participants who completed the survey.

### 3.6.6 Analysis

We follow a *grounded theory* approach to qualitatively analyze the results provided by the survey participants. A grounded theory approach involves progressive identification and integration of categories of concepts (e.g., datasets quality attributes and metrics) from data that leads to the construction of theories directly grounded in data. We use the classical grounded theory approach described by Glaser due to its emphasis on the emergence of concepts [76, 165], i.e., an inductive rather than a deductive process.

Open questions are analyzed using *open coding* to extract all relevant information to the quality of traceability datasets [96, 155]. Open coding in grounded theory method is the

process by which we generate concepts from the data which are going to be the building blocks for the theory [76,166]. Open coding is the analytic process by which *concepts* (codes) to the observed data and phenomenon are attached during qualitative data analysis. Open coding generates codes for discussions in the data that can be clustered into concepts and categories. During open coding, concepts are generated by asking generative questions such as “What is this data a study of?”, “What datasets quality category does this expert feedback indicate?”, “What dataset concerns and characteristics can be driven from the expert’s response?”

Open coding starts with the review of the data and focuses on identification of the concepts and generation of a descriptive tag for each answer. For instance, in the collected data, we are looking for extracting the key characteristics related to how researchers reason about and evaluate the quality of traceability datasets and the metrics that can be used to assess those characteristics.

We perform three iterations of the coding phase. In a first iteration, one author of this paper coded the participant’s responses for one of the survey questions and then reviews and discusses the outcome with the remaining authors of this paper. In a second iteration, this author coded all the remaining responses and took *memos* to be used by the other authors when checking periodically all coded responses. All the generated codes were reviewed and discussed by all the authors in regular coding review sessions.

Throughout this open coding process, we perform a *constant comparative analysis* in which we compared experts feedback as well as the codes associated to them with each other, in order to unify codes, identify variations in code and potentially emerge new codes.

Throughout this iterative process of open coding and constant comparative analysis, we capture our insights in memos [6,75]. Our memos mostly encompassed information on the rationale behind considering a code as dataset quality attribute.

Through open coding and constant comparative analysis, we grouped the codes into core categories each representing a quality attribute. All the authors participated in the review and the discussion of the coded responses and categories.

Multiple choice questions are analyzed using histograms, stacked bar charts, and box-plots. For both **RQ6** and **RQ7** we provide quantitative and qualitative analysis.

## 3.7 Goal 6: T-DQA Web-Tool:

A functional prototype of the automated approaches is developed and released as a web-based tool<sup>4</sup>. First, we have manually searched for and collected traceability datasets. Next, all T-DQA v2 metrics were applied to the datasets and made available for researchers online. The tool provides multiple filtration parameters based on the quality metrics that allow researchers to better choose the dataset that matches their needs. In the next sections, we will provide the details and of the collected datasets and illustrate the functionality of T-DQA web-tool.

### 3.7.1 Datasets collection

Datasets are the cornerstone for T-DQA web-tool. To make this tool useful for researchers and practitioners, we needed to have several datasets that are rich in traceability artifacts. In addition, they must vary in terms of the quality metrics to provide researchers with a wide selection of datasets that matches their needs and the problem on hand that they are

---

<sup>4</sup><http://design.se.rit.edu/T-DQA/>

addressing. There have been some efforts to collect and provide researchers with traceability datasets. Table 3.7, gives an overview of two online sources of traceability datasets that were cited by many researchers based on our literature review findings.

Table 3.7: Existing Traceability-Datasets Sources

COEST				
# of Datasets	Traceability Datasets	Domains	Consider Quality	Updated
15	15	Healthcare Aerospace Transportation Entertainment Software Engineering Office Automation	No	Yes
PROMISE				
20	2	Aerospace	No	No

As clearly shown in the above table, although the COEST website offers 15 traceability datasets that represent a variety of domains and are updated, it lacks the quality consideration that allows researchers to choose datasets based on their needs. On the other hand, we can see that the PROMISE website lacks a reasonable number of datasets where only two traceability datasets exist. Also, both datasets are representing a single domain which is the Aerospace domain.

To achieve our goal of having a large number of traceability datasets, we conducted an intensive manual search for such datasets not only on the previously mentioned resources (COEST and PROMISE) but also including the existing literature in Software traceability for the past fifteen years. All publications that have a traceability theme and used datasets in their experiments or tool evaluation were considered. Then we have searched for any existing online link for the datasets that were listed by the author. If we couldn't find any links or in the case of invalid links, we conduct an online search for the datasets in the well known OSS repositories such GitHub<sup>5</sup> and sourceforge<sup>6</sup>. In the case of failing to find the required datasets, a list of authors who have used these datasets was created to contact them asking for a valid link or sharing of the datasets.

Table 3.8, gives an overview of the traceability datasets that resulted from this intensive manual search and made available by our T-DQA web-tool to the researchers. As shown in the table, the web-tool offers a total of 37 traceability datasets that represents a variety of domains. T-DQA web-tool not only offers a larger number of datasets that by far exceed what existing resources offer but also takes into consideration quality metrics which is going to be covered in the following section. Table 3.9, list all the datasets in our web-tool and their corresponding domains.

### 3.7.2 T-DQA metrics

T-DQA web-tool implements the metrics of our previously quality framework that was described in Section 4.7.1(T-DQA v2). Table 3.10 Shows all the quality metrics that were

<sup>5</sup><https://github.com>

<sup>6</sup><https://sourceforge.net>



Table 3.8: T-DQA Web-Tool Traceability-Datasets Summary

<b>T-DQA Web-Tool</b>				
# of Datasets	Traceability Datasets	Domains	Consider Quality	Updated
37	37	Healthcare Aerospace Transportation Entertainment Software Engineering Office Automation Industrial Misc.	Yes	Yes

applied to the 37 datasets to assess their quality. These metrics were listed in T-DQA web-tool as filters to allow researchers to choose the dataset that matches their needs in an easy way while showing full details of the datasets. There are 15 metrics in total that represent four main quality dimensions Accessibility, Intrinsic, Contextual, and Representational. The diversity of the metrics and their dimensions provide the researchers with a big picture of the quality of the datasets and make it easier for them to decide which datasets are suitable for the problem they are tackling. The interface of the T-DQA web-tool is going to be described in the following section.

Table 3.9: List of T-DQA Web-Tool Datasets

Dataset Name	Domain	Dataset Name	Domain
Albergate	Office Automation	Pine	Software engineering tool
WV_CCHIT	Healthcare	WorldVistA	Healthcare
CM1-NASA	Aerospace	AgileOERP	Software engineering tool
eANCI	Office Automation	Modis	Aerospace
EasyClinic	Healthcare	RETRO.NET	Software engineering tool
Event Based Traceability (EBT)	Misc.	Aqualush Benchmark	Industrial
eTOUR	Entertainment	CCHIT	Healthcare
GANNT	Software engineering tool	Soren	Healthcare
IceBreaker	Transportation	Consultations	Healthcare
InfusionPump	Healthcare	Waterloo	Misc.
iTrust	Office Automation	iTrust2	Healthcare
Kiosk	Healthcare	PatientOS	Healthcare
SMOS	Transportation	PracticeOne	Healthcare
WARC	Software engineering tool	Trial Implementations	Healthcare
Apach Ant	Software engineering tool	WorldVistA2	Healthcare
ArgoUML	Software engineering tool	Care2x	Healthcare
Dependency Finder	Software engineering tool	ClearHealth	Healthcare
JHotDraw	Software engineering tool	Dronology	Aerospace
PURE	Software engineering tool		

Table 3.10: T-DQA Web-Tool Metrics

Metric Name	Type	Dimension
Domain	Categorical	Intrinsic
Type	Categorical	Intrinsic
Programming language	Categorical	Intrinsic
Popular	Binary	Intrinsic
Multi-Version	Binary	Intrinsic
Artifacts size	Numerical	Intrinsic
Artifacts type	Categorical	Contextual
Artifacts format	Categorical	Representational
Oracle is present	Binary	Intrinsic
Oracle developer	Categorical	Contextual
Oracle size	Numerical	Intrinsic
Oracle collection	Categorical	Contextual
Licensing	Binary	Accessibility
Storage	Categorical	Accessibility
Industry representative	Binary	Contextual

# Chapter 4

## Results

Currently, we have conducted the first and second steps from our research agenda. In the third step, we have conducted a limited study that considers only one traceability scenario. The main findings from our study to answer each research question are as follow:

### 4.1 RQ1: What are the characteristics of traceability datasets?

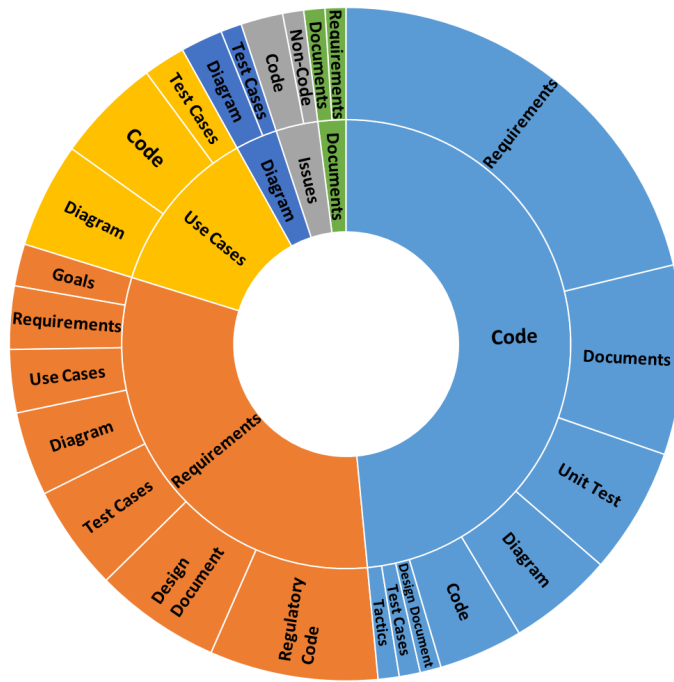
This question is investigated through five sub-questions described below. Each sub-question examines different characteristics of traceability datasets. From all the papers studied in our SLR study (78 papers), we identified 73 unique datasets.

#### 4.1.1 *RQ1.1: What are the source and target artifacts in traceability datasets?*

Figure 4.1 shows the types of artifacts covered by the 73 datasets. The inner layer represents the *source* artifact type and the outside layer represents the *target* artifact type. Artifacts from the inner and outside layers are colored identically when there is an association between them in the datasets. More details about these data points and their frequency can be found in our online appendix.

All artifacts that specify textual requirement documents related to a dataset such as high level, low level, functional, and non-functional requirements are grouped under the “Requirements” category. In a similar fashion, the “Code” category consists of Java Classes, Code, Methods, and Classes. The “Test Cases” category groups all non-code test documents whereas the “Unit test” category is composed of the actual code implementations. The category “Document” is composed of artifacts such as manuals and other pages that datasets are being traced to.

As per our analysis, the most frequently, datasets are used to study traces between Code to: Code (4), Unit Test (6) and other Non-Code artifacts such as Requirements (21),

Figure 4.1: Common *Source* and *Target* Artifacts.

Documents (9), Diagram (5), Design Document (1), Test Cases (1) and Tactics (1). Another category of commonly considered artifacts was found to be between “Requirements” and other Non-Code artifacts such as Design Document (6), Goals (2), Regulatory Code (8), Test Cases (5), Diagram (4), Requirements (3), and Use Cases (3). Less studied artifacts were Use Cases, Issue Reports, Diagrams, and Documents.

#### 4.1.2 RQ1.2: Which application domains are represented by traceability datasets?

The frequencies and domains of the datasets are shown as a heat-map in Figure 4.2. Each colored block refers to an application domain. The sub-areas within a block represent a particular dataset where the area represents the frequency of that dataset usage.

*Healthcare* is by far the most frequent domain for traceability datasets [9, 17, 18, 33–35, 40, 54, 55, 57–61, 71, 99, 105, 108–110, 112, 113, 136, 141, 142, 151, 161]. This is not surprising as traceability is crucial for safety critical and highly regulated domains [45]. Similarly, datasets from the *Aerospace* domain are frequently used by researchers [35, 58, 70, 85, 87–89, 99, 104, 108, 109, 136, 140, 141, 161, 167, 168, 177, 189]. High proportion of datasets are also from the domains of *software engineering tools* [8, 37, 49, 56, 67, 134, 146–148, 181, 182, 189], *development libraries* [8, 11, 12, 37, 49, 50, 57, 85, 99, 100, 119, 179], and *entertainment* [17, 18, 35, 46, 51, 63, 68, 71, 72, 85, 91, 102, 104, 109, 111–113, 136, 142, 144, 161]. The majority of these systems are open source and available online which might explain their frequent usage by researchers. In

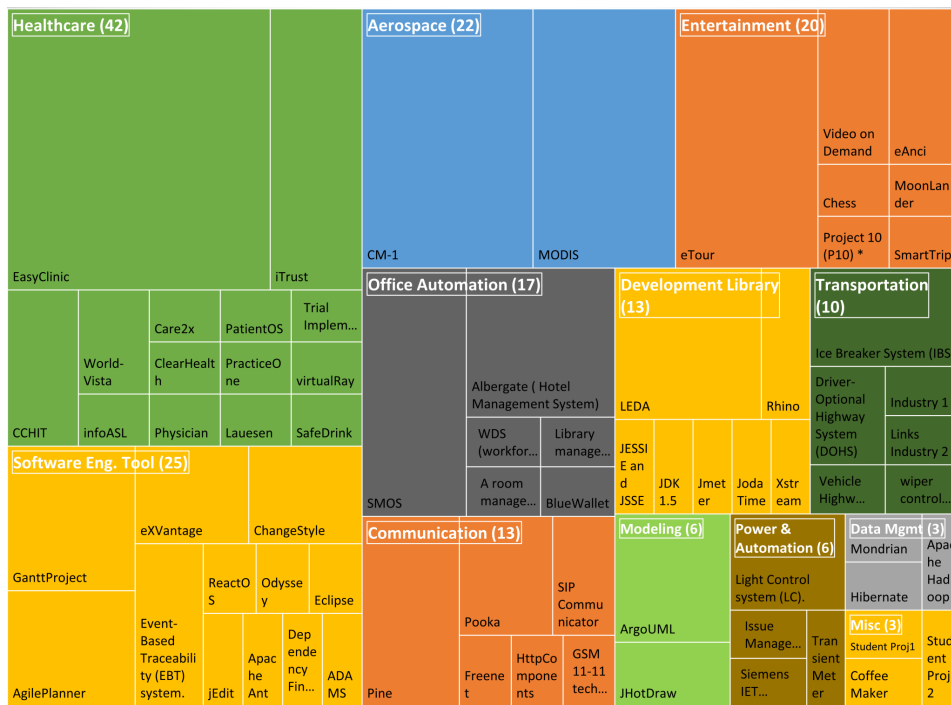


Figure 4.2: Dataset Domains and Frequency of Use.

addition, the researchers already serve as subject matter experts for some of these domains (e.g., software engineering tools or development libraries).

Industries such as *Power & Automation* have been used but less frequently [22, 38, 39, 62, 182, 189]. All except one of the datasets from this domain are closed source.

#### 4.1.3 RQ1.3: What is the size of traceability datasets?

Our analysis shows that there is an enormous gap in size among datasets that have been used by researchers (Table 4.1). The minimum trace space size is from the industrial datasets and it is 42 while the maximum one is over 29 million and it is a software system in the *power and automation* domain, containing 4845 issue reports and 6104 non-code artifacts [22]. The median of the trace space size among the three different datasets sources is relatively small.

Table 4.1: Datasets’ trace space statistics.

Statistics	OSS	Private/Industrial	University/Students
Minimum	264	42	50
First Quartile	870	1082	1515
Median	2028	2926	5135
Third Quartile	6956	131690	15472
Maximum	49810	29573880	390978

#### 4.1.4 RQ1.4: What proportion of the traceability datasets is from industry, open-source projects, and student generated data?

As shown in Figure 4.3 there is a fair distribution among the different types of sources: 31 datasets are open-source software (OSS), 24 datasets come from academia (e.g. student projects), and 18 datasets are industrial projects.

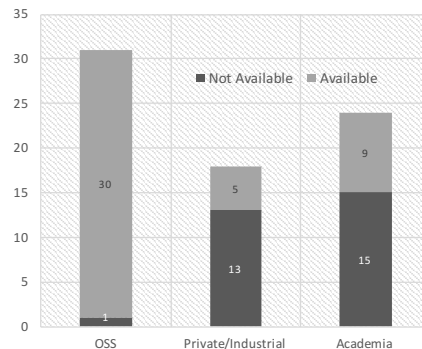


Figure 4.3: Source and availability of datasets.

#### 4.1.5 *RQ1.5: Are traceability datasets available for reuse?*

Figure 4.3 shows that 39.7% of the datasets (29 out of 73) are not available. Almost all of the OSS datasets are available. The majority of the industrial datasets (13 out of 18) are not available. The majority of the datasets coming from academia are not available (15 out of 24).

#### 4.1.6 *RQ1.6: Is there a relation between the characteristics and the quality of traceability datasets on the one hand and their reusability on the other hand?*

To answer this research question, we examined the relationship between reusability of a dataset (dependent variable) and the dataset’s quality metrics as depicted in column “Metrics” of Table 4.2 (independent variables). We selected all datasets for which we were able to retrieve or calculate the values for the quality metrics, i.e., 46 datasets.

Usability is a dichotomous variable, thus to compute the values for all datasets we encoded all datasets used at least twice as 1 and all datasets that are used only once as 0. We built a Random Forest importance plot [27] to determine the metrics that best predict the reusability of datasets.

Figure 4.4 indicates the importance of the quality metrics for the reusability of datasets. We observe that the most important metric is the team of *developers* that creates the dataset. This can be explained by the fact that almost all of the open source projects are available and thus facilitate reusability. *Size*, which corresponds to the number of artifacts in the dataset, appears to be an important factor as well. Other important factors are the *domain*, which is in accordance with the discussion in section RQ1.2, and the completeness of the datasets in terms of *source and target* artifacts.

Additionally, we built a linear regression model with the *actual frequency of use* as dependent variable. When performing multivariate regression we must account for possible risk of multicollinearity (i.e., interaction among the independent variables). A common way to deal with multicollinearity is to compute the Variance Inflation Factors (VIF) for each independent variable in the regression model and retain only those with low values—e.g.,  $\leq 2.5$  [36, 160]. After removing independent variables and non-significant variables, the only remaining independent variable is AnswerSet with coefficient 6.8352 (p-value=2.67e-07). The percentage of variance of the data explained by the model is about 45%.

#### 4.1.7 *RQ1.7: What are the threats to validity associated with traceability datasets?*

Among the 78 papers included in our SLR, 40% did not include a section related to the threats to validity nor discussed such concern while heavily relying on datasets to make research conclusion. 6% of the papers did have a threats to validity section, but did not identify any threats related to the usage of the data in their studies. Lastly, 54% of the papers discussed the threats to validity of their research related to the usage of datasets.

Two of the authors extracted all the threats to validity related to the datasets and manually grouped them. Note that we include all threats that are discussed by the authors of the respective papers which means that they were not necessarily mitigated. The threats to validity are as follows:



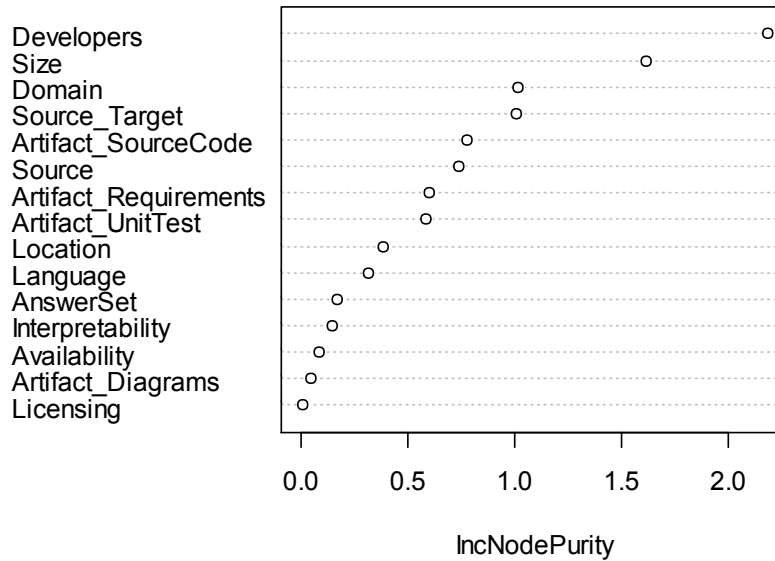


Figure 4.4: Random Forest Importance Variable Plot.

- *Trustworthiness*
  - *Artificial AnswerSet*: This threat is concerned with how answersets are created [37, 54, 82, 89, 108, 110, 127, 149]. Often the trustworthiness threat is not mitigated as the answersets are established by students rather than the original developers.
  - *Students Dataset*: This threat concerns dataset that are developed by students [71].
  - *Vetting Datasets*: This threat concerns datasets, particularly answersets, that are not vetted nor peer-reviewed [89, 127].
- *Threats to external validity*
  - *Real-World Data*: This threat is concerned with whether the datasets are representative of industrial projects [7, 9, 17, 18, 35, 37, 40, 54, 61, 63, 67, 71, 89, 91, 104, 109, 113, 114, 124, 127, 136, 140, 142].
  - *Limited Observations*: This threat is concerned with whether a limited number of case studies are used to validate the results [7, 9, 50, 59, 70, 81, 89, 91, 99, 108, 110, 113, 127, 140, 141, 189].
  - *Domain*: This threat is a concern when all datasets belong to the same application domain [37, 81, 82, 104, 124, 151] or when the number of datasets is insufficient to generalize the conclusions for a particular domain.

- *Cross Industry*: When an industrial dataset is used, this threat is concerned with whether the results are applicable to other industrial systems [137].
  - *Size*: This threat is related to the small size of datasets, impacting the generalizability of the results [8, 9, 17, 35, 54, 67, 89, 91, 104, 109, 112, 124, 140, 147, 151].
  - *Programming Language*: This threat is a concern when datasets are in a specific programming language [33, 34, 71].
  - *Artifact Type*: This threat is concerned with the diversity of the type of artifacts available for the datasets (e.g., requirements, test cases, etc.) [99, 151].
- *Data Acquisition*
    - *Selection bias*: When datasets are not representative of the intended population (cherry picking) [40] or do not fit the problem [26, 40, 63, 70, 102]. For instance, this happens when a dataset from a non-safety critical project is used for a safety critical research study.
    - *Dataset-Equivalency*: This threat to validity concerns cases where researchers compare certain characteristics of their datasets with datasets used by previous researchers to justify the adequacy of the selected datasets [9, 63].
    - *Information bias*: Accuracy of the automatically generated datasets; misclassification and labeling of the data to be used [33, 33, 34, 102].
    - *Negative Set Bias*: Rich and unbiased selection of negative cases in training data, a common threat in classification problems [127].

#### 4.1.8 RQ1.8: Do we, as a community, strive for a diversity of traceability datasets?

To answer this research question, we studied the diversity of datasets used by authors across different research papers. First, we identified all authors who have published more than one traceability paper. This took us from 128 authors served on the 78 studied papers to 38 authors who have published more than one paper. For each of these authors, we calculated the diversity metric defined in Equation 3.2.

Figure 4.5 shows the results. The X-axis represents the total number of datasets used by each author. The Y-axis represents the total number of papers from each author in this SLR. The Z-axis represents the diversity ratio for the datasets used by the authors. Each vertical drop-line corresponds to one of the 38 authors. 12 authors from these 38, have a diversity ratio of 70% and above, 27 authors have a diversity rate of 50% and above, and lastly, 11 authors have a diversity rate below 50%.

We observe that in general authors with low number of datasets and low number of papers have a higher diversity ratio. One of the authors with high number of datasets and high number of papers has a high diversity rate. This example highlights individual effort in seeking diverse datasets for development and evaluation of various traceability solutions.

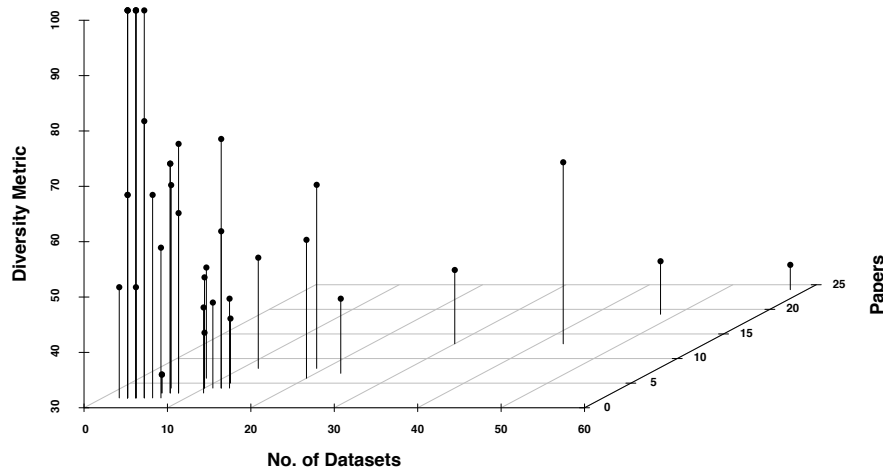


Figure 4.5: Authors and their Dataset Diversity.

## 4.2 RQ2: How to assess the quality of traceability datasets?

Data Quality Assessment frameworks are adopted in various fields of computing such as requirement engineering [53], information systems [101] [117], web linked data [184], data-warehousing [92] [120], and health-care [139], [94] to assess the quality of scientific datasets. Some of these frameworks are specific to a domain, while others are applicable to a broad range of scientific datasets [174]. We have studied several data quality frameworks and we have adapted them to the domain of software traceability. We propose a Traceability-Dataset Quality Assessment (T-DQA) framework that is suited for assessing datasets in the field of traceability.

Following the approach of Wang et al. [174] we broadly classified the T-DQA under four main quality dimensions: *intrinsic*, *contextual*, *representational*, and *accessibility*. These in turn comprise of 10 sub-dimensions and 13 quality metrics. Table 4.2 provides a summary of the proposed T-DQA framework with the dimensions, sub-dimensions, definitions, metrics definitions and types. The metrics are adapted from the existing literature on assessment of data quality in a broad sense [174] and from the domain of *linked data* which to some extent is similar to traceability [184].

To illustrate the different dimensions in the T-DQA framework, we use the Easy-Clinic dataset as an example. *Easy-Clinic* implements all operations required to manage a medical ambulatory.

- The *accessibility* dimension accounts for data concerns related to access, authenticity, and retrieval [184]. The EasyClinic dataset is publicly available on the COEST GitHub repository under the General Public License.
- The *intrinsic* dimension captures the characteristics of the dataset that are inherent

Table 4.2: Traceability-Dataset Quality Assessment (T-DQA) Framework: dimensions, and definitions.

Dimensions	Sub - Dimensions	Definition	Metrics	Metric Type
Accessibility	Availability	Availability of a dataset is the extent to which data is present, obtainable and ready for use.	Dataset can be downloaded from the link provided.	Binary
	Licensing	Licensing is defined as the granting of permission for a consumer to re-use a dataset under defined conditions.	license agreement exists.	Binary
	Storage	Where is the Dataset Stored?	Private Server/ Organizational Server /Public	Categorical
Intrinsic	Domain	A specified sphere of activity or knowledge having common set of requirements, terminology, and functionality	Application Domain	Categorical
	Completeness	Completeness refers to the degree to which all required information is present in a particular dataset.	Source-Target artifacts are present. (at least 2 types of artifacts are present in dataset) Answer Set is Present	Binary Binary
	Developers	Team responsible for creating the dataset.	Open source community, industrial or academic	Categorical
	Programming Language	A programming language is a formal computer language designed to communicate instructions to a machine.	Java/ C++ etc.	Categorical
Contextual	Relevancy	Relevancy refers to the provision of information which is in accordance with the task at hand and important to the users' query.	Size – Total number of artifacts Number of artifacts for each specific type (Req., UML Diagrams, Code, Test, and etc.)	Numerical (Type, Numerical)
	Trustworthiness	Trustworthiness is defined as the degree to which the information is accepted to be correct, true, real, and credible.	Dataset Source Frequency of Usage	Categorical Numerical
Representational	Interpretability	It refers to technical aspects of the data, that is, whether information is represented using an appropriate notation and whether the machine is able to process the data.	detecting the use of appropriate language, symbols, units, datatypes and clear definitions	Categorical

to itself and independent of its usage. Characteristics such as domain, completeness of dataset, developers (e.g. open source vs. industrial) and programming languages can be used as intrinsic quality indicators helping researchers to reason about the suitability of a dataset for a research problem. For instance, Easy-Clinic belongs to the HealthCare domain. It is considered complete for use in traceability as it consists of at least two artifacts (code, requirements, etc.) and it contains an answerset. It was developed by Master students at the University of Salerno. It is written in Java.

- The *contextual* dimension captures how suitable a dataset is in a particular research context. While a dataset might be good for tracing requirements to source code, it might not be suitable for tracing requirements to tests or to design documents. Contextual quality indicators are the relevancy of a dataset to a research problem and the trustworthiness of the dataset in that context. Frequently used datasets for evaluating a particular research problem are typically considered as benchmark data, which adds to the reputability of the datasets, and facilitates the comparison of the results across different papers [174]. Easy-Clinic consists of 30 use cases, 20 interaction diagrams, 63 test cases, and 37 code classes, accounting for a total of 160 artifacts and an answerset of 1005 trace links. The number of artifacts are contextual metrics as their usage is relative to the traceability task at hand. For instance, in case of requirement-to-requirement traceability, datasets having higher number of requirement artifacts such as CM1 (235 High Level X 220 Low Level) and MODIS (19 High Level X 49 Low level) are better suited than others that have lower number of requirement artifacts. Easy-Clinic is a student project. Its reuse factor is 20 as it is used as dataset in 20 papers.
- The *representational* dimension concerns the format in which a dataset is available. The quality of a dataset depends on how the dataset is packed and shared with

others. This metric is indicative of the dataset formats such as XML, PDF, Word, Source-Code. EasyClinic is available on COEST website in XML format.

This framework helps researchers to be conscious of the different dimensions that need to be considered when choosing a dataset for a traceability research problem at hand.

### 4.3 RQ3: Is it feasible to automatically generate datasets from open source software repositories?

The experiments design described in section 3.4.8 was followed to train the tactic classifier using three baseline approaches and compare the results. Table 4.3 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 tactical file approaches unsurprisingly learned more code-oriented terms such as *ping*, *isonlin*, and *pwriter*.

Figure 4.6 reports the F-Measure results for classifying classes by tactic using several combinations of threshold values. Overall three baseline methods obtained similar accuracy. In two cases, namely *audit* and *heartbeat* the classifier trained using expert-collected files 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 the Web-Mining-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.

Table 4.4 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 Big-Data-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.

#### 4.3.1 RQ3.1: Does the training method based on automated web-mining result in higher trace-links classification accuracy compared to an expert-created training set?

The above results indicate that, in four out of five cases the manual expert-created approach outperformed the automated Web-Mining technique. However, the differences were very small. Table 4.5 shows the differences between the F-Measure of the manual expert-created and the automated Web-Mining approaches.

Based on this limited observation, we can rank the manual expert-created baseline method equivalent to the automated Web-Mining approach. In order to evaluate whether the differences were statistically significant we performed *Wilcoxon* tests as well as the *Friedman*

Table 4.3: Indicator terms learned during training

<b>Tactic Name</b>	<b>Web-Mining trained indicator terms</b>	<b>Big-Data trained indicator terms</b>	<b>expert-created trained indicator terms</b>
<b>Heartbeat</b>	nlb cluster balanc wlb ip unicast net- work subnet heartbeat host	counter, fd, hb, heartbeat, member, mbr, suspect, ping, hdr, shun	heartbeat, ping, beat, heart, hb, out- bound, puls, hsr, period, isonlin
<b>Scheduling</b>	schedul paral- lel task queue partition thread unord ppl concurr unobserv	schedul, pri- oriti, task, feasibl, prio, norm, con- sid, paramet, polici, thread	schedul, task, prioriti, prcb, sched, thread, rtp, weight, tsi
<b>Authentication</b>	authent, pass- word, user, account, cre- denti, login, membership, access, server, sql	password, login, user- nam, rememb, form, authent, persist, sign, panel, succeed	authent, credenti, chal- leng, kerbero, auth, login, otp, cred, share, sasl
<b>Resource Pooling</b>	thread, wait, pool, ap- plic, perform, server, net, ob- ject, memori, worker	pool, job, thread, con- nect, idl, anonym, async, context, suspend, ms	pool, thread, connect, spar- row, nbp, pro- cessor, worker, timewait, jdbc, ti
<b>Audit Trail</b>	audit, trans- act, log, sql, server, secur, net, applic, databas, manag	trail, audit, categori, ob- serv, udit, outcom, ix, bject, acso, lesser	audit, trail, wizard, pwriter, lthread, log, string, cate- gori, pstmt, pmr

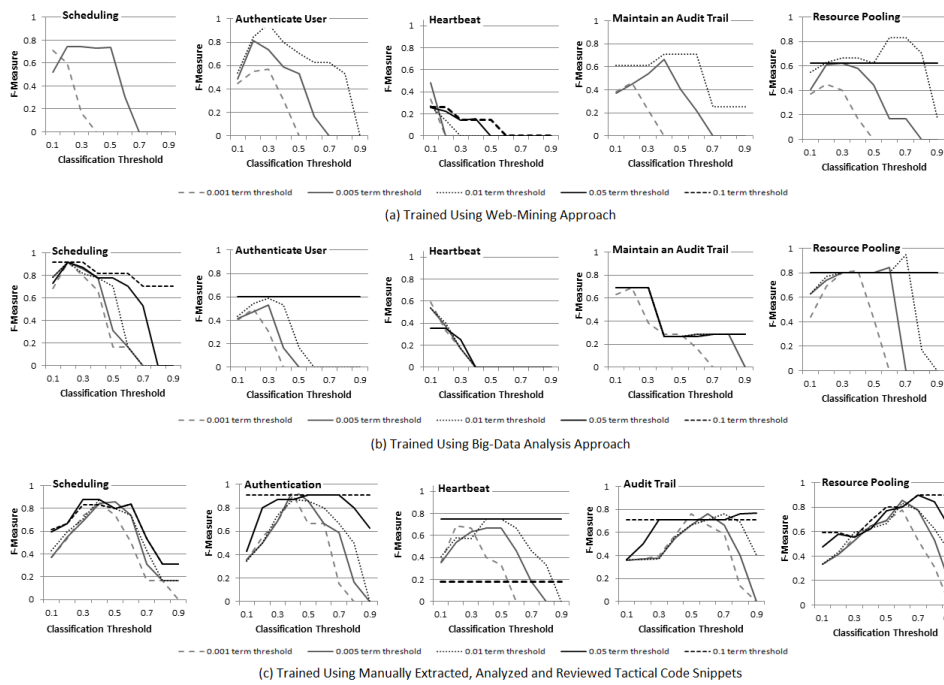


Figure 4.6: Results for Detection of Tactic-related Classes at various Classification and Term Thresholds for five Different Tactics

Table 4.4: A Summary of the Highest Scoring Results

Tactic	Training Method	FMeasure	Recall	Prec.	Spec.	Term/ Classification threshold
<b>Audit</b>	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-Created	0.758	1	0.611	0.833	0.001 / 0.5
<b>Authentication</b>	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-Created	0.956	1	0.916	0.977	0.005 / 0.4
<b>Heartbeat</b>	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-Created	0.689	1	0.526	0.775	0.001 / 0.2
<b>Pooling</b>	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-Created	0.9	0.818	1	1	0.05 / 0.7
<b>Scheduling</b>	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-Created	0.88	1	0.785	0.931	0.01 / 0.4

Table 4.5: Differences in F-Measure of the Expert-Created and Web-Mining Approaches

Audit	Authenticate	Heartbeat	Pooling	Scheduling
0.048	0	0.209	0.067	0.14



ANOVA test which is a non-parametric test for comparing 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.)<sup>1</sup>

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

### 4.3.2 RQ3.2: Does the training method based on automated big-data result in higher trace-links classification accuracy compared to an expert-created training set?

Table 4.6 shows the differences between the F-Measure of two approaches (Subtract expert-created F-Measure from Big-Data F-Measure ). As we can see in the table for the classification of two out of the five tactics (Scheduling and Pooling), the Big-Data Analysis method outperformed the manual expert-created approach. In two of the remaining cases (Heartbeat and Audit), both methods returned very close results.

Table 4.6: Differences in F-Measure of manual expert-created 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. <sup>2</sup>

**Result :** There is no statistically significant difference between the trace link classification accuracy for a classifier trained using expert-created 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.

### 4.3.3 RQ3.3: What is the impact of training set size on the accuracy of trace link classification?

An ongoing debate exists on the research techniques examined/developed using student-generated datasets. The community has utilized different mitigation techniques to minimize the biases and threats related to this set of approaches [74, 107]. At the same time, the

<sup>1</sup>p-value of 0.05

<sup>2</sup>p-value of 0.05

community has praised the notion of an Expert-Created approach for obtaining datasets. 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. It is commonly perceived that the larger the size of training set, the more accurate and generalizable the underlying 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. Based on this, and our research question RQ3.3, we make the following **Null Hypothesis**: *a larger training-set size will not result in a more accurate and generalizable learning method.*

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, especially because such an extension can have a significant cost. With all the mitigation techniques used to minimize the threat to validity and create generalizable training sets, 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 the 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.

In our very first work in this area [131], we used training sets of files from 10 software systems. In extension of this work [126] we used files sampled through a peer-reviewed process from 50 open source projects. Considering that the training sets were established using systematic manual peer review, extending them from 10 open source projects to 50 projects took almost 6 additional months of work. The experiment described in the next sub-section aims to investigate the increase in accuracy of classifiers for such additional cost.

**Experiment design:** 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 (random sampling of tactical files) 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.

For this investigation we use two software systems outside of our original dataset, Apache Hadoop and OfBiz. These two projects are widely used in industry and are representative of complex software systems. We made sure, that these two projects are not part of 50 projects used in the training set. So there was no overlap between training data and the case studies used as the testing set. First the classifiers were trained using 5 randomly-selected sub-samples of this dataset in the size ranges of 10, 20, 30, 40, and 50 sample files. Then, each classifier was used against the source code of the two projects.

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

The accuracy metrics are reported in figure 4.7. The bars in this graph show precision, recall and specificity [131]. 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.

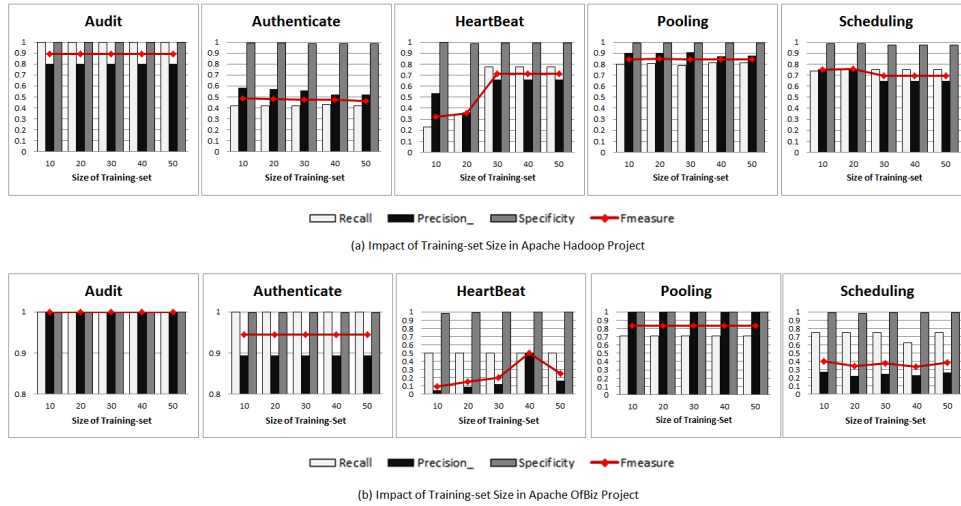


Figure 4.7: The impact of training-set size in manually established dataset on accuracy of recovering trace links

**Result :** This observation supports the notion that in case of manually creating a high quality training set, the size of the dataset will not have a significant impact on the accuracy of the classification technique described in equations 3.4 and 3.3. Collecting of 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 these two case studies. In future works, we will run more experiments, to investigate if this would be valid across different systems.

#### 4.3.4 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. The primary cost for automated techniques is query formation. However, this cost is significantly minimal compared to the manual search for the artifacts. For instance, in our experiments each query was formed by going through the definition of the tactic, and it took approximately less than 2 minutes to finalize the query.

**Cost-Comparison** In an earlier work, the estimated cost (in terms of time) for creating the training set using the expert-created 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-Data approach generates a similar file 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 shown in Table 4.7. Overall, the automated approach based on Big-Data analysis has created more correct data points (tactical files) 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 4.7: Accuracy 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 files from our large scale software repository, we observed that the data collected by experts exhibits higher internal quality. The manually collected training-set not only contains 100% accurate data points (due to 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 files 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 files, they will also take into account metrics related to data quality and sampling strategies.

### 4.3.5 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)<sup>3</sup>. 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 4.8 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

<sup>3</sup><http://design.se.rit.edu/budget/>

and negative samples) and the datasets can be automatically created using both Web-Mining and Big Data analysis techniques or only one.

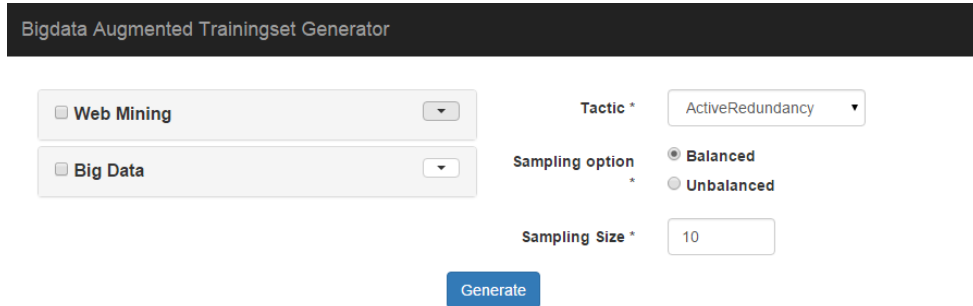


Figure 4.8: User interface for selecting the data generation parameters of BUDGET tool

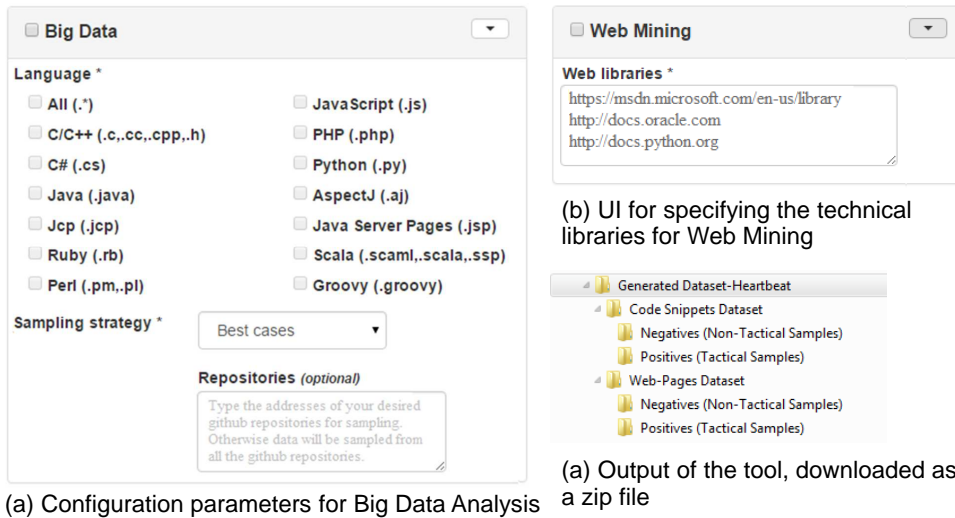


Figure 4.9: Configuration parameters for Big Data Analysis, Web-Mining Approach and Generated Output

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 4.9 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 tactic-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 tactic-query is modified to only return web pages that do not contain any of the tactic-related terms.

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 116,609 projects, continuously more open source projects are being added to this repository.

BUDGET's parameter for generating training set of tactical files include programming languages of the source codes, and a sampling strategy (Figure 4.9). The sampling strategy defines how BUDGET should sample the tactic-related files 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 4.9 shows the folder hierarchy of the datasets generated.

### 4.3.6 Discussions

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-created 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-created dataset used in investigating the impact of dataset size is also released on-line at COEST.org.<sup>4</sup>

### 4.3.7 Generalization of results to other classification techniques

Throughout this work we used our own custom-made classification technique. This was primarily done because the previous work by M. Mirakhorli et al [125] shows that our tactic classifiers outperform off-the-shelf classifiers in identifying tactical files. In this section we investigate how another baseline classifier will perform using the data automatically generated by our approach.

To do so, we repeated the experiments in section 3.4.8 using a Naïve classifier.<sup>5</sup> Table 4.8 reports the difference between F-Measure of Expert-Created and Big-Data Analysis approaches. In case of Scheduling tactic, the Naïve classifier trained using Big-Data Analysis approach outperformed the Naïve classifier trained using the expert-created dataset. In case of Authentication tactic, both classifiers achieved similar F-Measure. In case of Audit the F-Measure of both approach was very close while in the remaining two tactics (Heartbeat and Pooling) the expert-created dataset resulted in better F-Measure.

Table 4.8: Differences in F-Measure of Expert-Created and Big-Data Analysis Approach in Naïve Bayes Approach

Audit	Authenticate	Heartbeat	Pooling	Scheduling
0.038	0	0.179	0.185	-0.21

These results are inline with our observation of how our custom-made classifier performed in the experiment described in the section 4.3. Similarly, in order to evaluate whether differences were statistically significant we performed *Wilcoxon* tests as well as the *Friedman ANOVA*. In both statistical tests, we could not reject the null hypothesis that there is a difference in median/mean-rank of two groups.<sup>6</sup>

<sup>4</sup><http://coest.org/mt/27/150>

<sup>5</sup>Weka's NaiveBayesMultinomialText method was used.

<sup>6</sup>p-value of 0.05

Table 4.9: Differences in F-Measure of Expert-Created and Web-Mining Approach in Naïve Bayes Approach

Audit	Authenticate	Heartbeat	Pooling	Scheduling
0.187	0.184	0.364	0.368	0.086

While we used Web-Mining approach to train the Naïve classifier, the F-Measure values obtained were lower than the values obtained when this classifier was trained using expert-created datasets. Similar statistical test were performed (*Wilcoxon* and *Friedman ANOVA*) and the results indicate that Naïve classifier while trained using expert-created dataset outperforms the Web-Mining technique.

Our inspection of this issue, indicates that Web-Based dataset has more diverse terminologies as well as noises compared to the source files extracted using Big-Data analysis approach. An appropriate feature selection algorithms [132] can help identify key discriminating terms in the training set.

In previous work [125] researchers has shown that custom tactic-classifier outperforms off-the-shelf classifiers. The key factor is the way our classification technique identifies indicator terms, that takes into account the impact of domain terms. The results obtain in this section, provide support that both Big-Data Analysis and Web-Mining approaches are as effective as Expert-Based approach when the data is collected for the tactic-classifier technique as shown in previous work by M. Mirakhorli et al [131]. Furthermore, Big-Data Analysis approach is as effective as Expert-Based approach when Naïve classifier is used. Considering the empirical results we obtained in an earlier work [125] that recognizes the tactic-classifier as best technique to identify tactical files, we can conclude that combined use of our automated dataset generation and tactic-classifier technique can result in the most cost effective way to create traceability datasets and trace tactics to the source code.

### 4.3.8 Qualitative insights

The quantitative experiments reported in previous sections provide evidence that automated techniques can be used to help researchers obtain datasets of software artifacts. In order to gain further insight into how these automated techniques work, and how datasets generated this way differentiate from expert-created datasets, we present a qualitative study in this section. We first compare the automatically generated datasets to the expert-created datasets from various perspectives. Then we analyze random samples of false positives and false negatives for each technique as well as cases reported as true positive for one technique and false positive/negative for another technique.

#### Datasets comparison

In section 4.3.4 we evaluated the quality of datasets generated using the Web-Mining and Big-Data Analysis approaches. This evaluation focused on the correctness of the items in the datasets. In this section, we compare the content of datasets generated using automated techniques with those obtained by the expert. Our comparison shows that the tactical source files labeled by the experts are richer in terms of terminology and they tend to have



more tactic related terms than those generated automatically. The automated approach normalizes the term frequencies over the the source file length, so most labeled files were relatively small. Comparatively, we found that the experts sometimes included very big source files which also contained a diverse set of tactical terms. Overall, the data generated automatically needed more context about the source file to understand how it related to a tactic, while source files labeled by experts were easier to understand. From the perspective of training a classifier, such differences can be less of an issue, however we plan to expand our approach and include files which are not only representative of a tactic, but also expose qualities similar to those identified by experts. To do this we will develop an algorithm which samples the source files based on their centrality to the tactic's implementation, or retrieves all the files involved in the implementation of the tactic. Our tool will be expanded to enable both of these sampling strategies.

### Comparison of classification features

In the next qualitative investigation we looked into the indicator terms identified from the expert-created and automatically generated datasets. The top ten indicator terms are depicted in table 4.3. While there are several commonalities between top 10 terms, we observed that among the top 30 terms, the Big-Data Analysis dataset contained more diverse terminologies. The Big-Data Analysis datasets contain other terms associated with the tactic which are not found in the expert-created dataset. As an example, for heartbeat the Big-Data Analysis data contains terms such as *Pinger, live, monitor, msg, health, timeout, ack, failure, heard, master, timer*. However these terms are not used during the classification process, because their score is smaller than indicator term threshold. This indicate a potential for augmenting the automated approach with other techniques such as Natural Language Processing (NLP) and Information Retrieval (IR) to better identify indicator terms.

This would enhance the generalizability of a trained classifier, so that it can identify tactics which are implemented using different frameworks and terminologies.

### Comparison of results

For each tactic we inspected a random misclassified case (true negatives, false positive/negative). The results show that in most of these cases a source file was misclassified as false positive because one or two of the indicator terms occurred within that source file. For instance, in case of Audit, the term Audit existed in one of non-tactic related files and that file was always classified as Audit Tactic, despite the fact that other relevant indicator terms such as (Trail, log, records) were missing. We believe that this can be improved by extending the classification equation to take into account the context in which the indicator terms appear as well as reducing the sensitivity of the classifier to a single feature. In cases of false negatives, the source file contained more of indicator terms with lower score (formula 3.3), therefore the classification score was below the chosen classification thresholds. We could not identify specific reasons for why different results was obtained by training the classifier over automated and manually generated datasets. The results primarily depends on the indicator terms discovered in the training phase and which of these terms occur in the testing datasets.

### 4.3.9 Application to the other Areas 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. Here we aim to conduct a feasibility study on using the proposed automated dataset creation technique to support research in different areas of requirements engineering.

In an initial study of resources in technical libraries such as MSDN as well as open source repositories, we identified artifact types which can easily be automatically generated using our approach. Therefore in the following sections we report traceability scenarios where these artifacts can be used.

#### Usage scenario#1: tracing regulatory codes.

We now present 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 CO-EST.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 [19]. For example, in Table 4.10 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. Three independent traceability researchers have evaluated the accuracy of the extracted data. The results are presented in Table 4.11 and indicated that 63% of automatically generated data points were correct. These are the extracted files while related to the search query used by our approach. The traceability researchers through a peer-review process evaluated each individual artifact and inspected whether the artifact is about Database Security concerns in HIPA or not. Here we provide an excerpt of two sample data points extracted using our approach:

- *“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....”*

Table 4.10: Sample Regulations Discussed on Technical Libraries

<b>ACT Name</b>	<b>Applies to</b>
<b>Sarbanes Oxley Act</b>	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 [19]. More on ( <a href="http://www.sec.gov/">http://www.sec.gov/</a> )
<b>HIPAA</b>	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. [19]
<b>PCI</b>	Payment Card Industry Data Security Standard (PCI DSS) is a proprietary information security regulation for organizations that handle branded credit cards. [4]
<b>The Gramm-LeachBliley Act (GLBA)</b>	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]
<b>SB 1386</b>	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]
<b>BASEL II</b>	Is recommendations on banking laws and regulations issued by the Basel Committee on Banking Supervision. Applies 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 [19].
<b>Health Level Seven (HL7)</b>	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 [19].

In the area automatically generating dataset precision or accuracy of generated dataset is more important than recall (retrieving all the artifacts from web). In our baseline Big-Data Analysis and Data-Mining approaches we have not utilized any tweaking to improve the precision of artifact retrieval. Common NLP techniques can result in greater artifact accuracy. Our study reported here, provides the accuracy metric of a simplified search technique. This is adequate to support the fact that automated dataset generation techniques will work and to draw the community attention to this research area. In future, work we plan to enhance the accuracy of our artifact retrieval techniques using NLP.

#### 4.3.10 Usage Scenario#2: Classifying Functional Requirements:

Another area where automated techniques can be used is the generation of datasets for classifying functional requirements. Classification in this context is primarily for tracibility purposes and to distinguish functional and nonfunctional requirements.

**Problem:** Traditionally the VSM [71] 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 [10]. The biggest drawback for this approach is the difficulty of obtaining several samples of the same functional requirements or files.

**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 files 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 4.11 shows the accuracy of the Big-Data analysis in establishing datasets for files implementing functional requirements of an ERP (Enterprise Resource Planning) software system. In fact, in all cases, our approach successfully created datasets of files 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 <sup>7</sup>. Although this is a feasibility study, it highlights the fact that our approach can be applicable to the other types of datasets beyond tactical-files. The purpose of our two usage scenario reported in this section is to show the potential for extending the automated dataset generation techniques to the other areas of software traceability. The positive results reported in the table 4.11 indicate that in presence of large corpus, information retrieval techniques can assist developers in obtaining the datasets which require extensive human involvement.

#### 4.3.11 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. 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-created dataset used in investigating the impact of dataset size is also released on-line at COEST.org.

**External validity** evaluates the generalizability of the results. One of the primary threats is related to the construction of the datasets for this study. The manual dataset

---

<sup>7</sup>Please see terms in the figures: [http://www.1tech.eu/clients/casestudy\\_ventraq](http://www.1tech.eu/clients/casestudy_ventraq)

included over 250 samples of tactic-related code. The task of locating and retrieving these files 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 files 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. On the other hand, the majority of identified keyterms are non-language specific. Therefore, the experimental results reported in this paper will not be impacted by this language-specific keyterms.

Furthermore, the Hadoop and OfBiz case studies were used to evaluate the impact of dataset size on accuracy of tactic classification on a large and realistic system. We recognize that these are only two case studies and cannot be representative of all typical software engineering environments. In future work, we will to explore additional case studies in an effort to generalize our observations.

**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. An additional threat in this category is that the accuracy of the automated techniques can be dependent on the tactic-query used in the study. To avoid this bias, we pre-selected 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.

#### 4.4 RQ4: Can we automatically detect and categorize open-source software artifacts?

We extracted 91,108 open-source projects in various programming languages from GitHub between April and October 2015. To achieve 95% confidence level and 5% margin of error, we randomly select 383 applications and study software artifacts in those projects. The size of the selected subjects, in terms of Lines Of Code (LOC), ranges from 2 to 12 million LOC.

Figure 4.10 shows the distribution of the primary programming language across the projects, i.e., the language with the highest number of LOC.

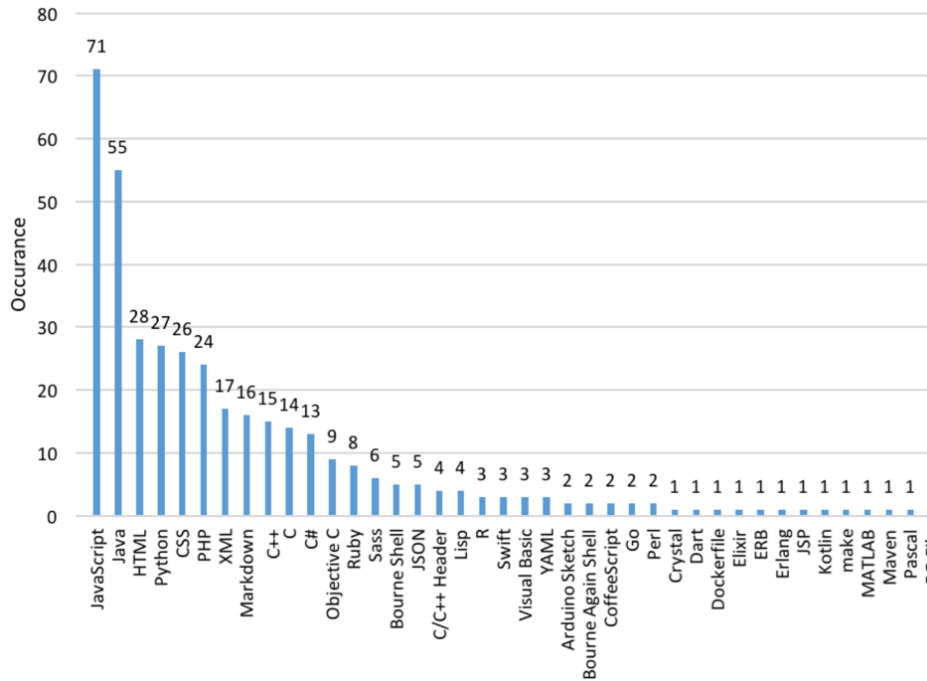


Figure 4.10: Distribution of primary languages in the sampled projects.

#### 4.4.1 RQ4.1: How can software artifacts be categorized?

We identify the following artifact types only by file names and extensions as shown in Table 4.12: application, archive, audio, disk image, font, image, project, source code, testing code, and miscellaneous. Some file extensions can be associated with multiple file types. For example, png can be Portable Network Graphics Image or Corel Paint Shop Pro Browser Catalogue, i.e., an image file or a documentation file. We randomly sample 5 instances of such extensions and assign them to one file type based on their file content. In addition to extensions we separate testing code from source code, by verifying if one of the following keywords appears in file name or directory: “test”, “tests”, and “mock”. Another heuristic is used to identify miscellaneous files based on the number of words in the file. Through experiments, we observe that a threshold of 30 offers a good compromise between precision and recall.

We analyze the file extensions associated with open-source projects. There are 234,296 artifacts with 1,217 distinct files extensions in the sampled projects, excluding hidden files.

Table 4.11: 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%

Table 4.12: Heuristics applied to identify types of non-documentation related artifacts.

Artifact Type	Heuristic
Application	.bat .cmd .exe .ser .swf
Archive	.a .gz .jar .pack .zip
Audio	.kt .mp3 .ogg .wav
Disk Image	.scl
Font	.eot .otf .ttf .woff
Image	.blp .bmp .dds .gif .ico .jpeg .jpg .png .psd .rs .svg .tga .tif .xpm
Project	.csproj .pbxproj .vcproj .vcxproj
Source Code	.as .asm .c .cc .class .coffee .cpp .cs .cshtml .css .ctp .cxx .d .dll .ebuild .ejs .el .erb .erl .f .f90 .go .gradle .groovy .h .haml .hpp .hs .i .java .js .jsp .less .lua .m .mo .o .php .phpt .phtml .pl .pm .pp .py .pyc .r .rb .s .scala .scss .scssc .sh .smali .so .sql .swift .t .tcl .ts .vb .vim .rkt
Testing Code	if a file is classified as code, we further examine if “test”, “tests”, and/or “mock” is contained in fully qualified file name, ex. ProjectName/src/test/file.java
Miscellaneous	non-readable files non-English files insufficient information (files with $\leq 30$ words)

However, the top 38 most frequent file extensions occur in more than 95% of the projects and account for over 76% of the total artifacts. Table 4.13 shows the top 38 most frequent file extensions along with the number of projects that contain files with these extensions and the number of files with these extensions in the sampled projects. “Num. of Projects” (%) reports on the number (percentage) of sampled applications that contain files of each extension. “Num. of Files” (%) reports on the number (percentage) of files with each extension in the sampled applications. “Cum. %” reports the cumulative % of the artifacts. For instance, the first row shows that 1) 383 out of the 383 sampled projects, i.e., 100%, contain files without extension and 2) 13,706 out of 234,296 artifacts, i.e., 5.85%, have no extension. The extensions highlighted in gray are documentation related files that are not identified by the heuristics shown in Table 4.12. Since it is not feasible to manually go through every single file, we sampled 2% of files with the highlighted extensions.

To create an oracle of documentation related files, two coders manually and independently classify 894 randomly selected artifacts. 149 out of 894 sampled artifacts are documentation related files. During this manual classification process, we iteratively refine and consolidate the initial list of categories as needed. The initial IRR value is 0.64 and it is calculated for a set of 115 artifacts. The two coders then discuss the discrepancies to reach an agreement. The subsequent IRR value increased to 0.786 for the next 115 artifacts, which indicates substantial agreement [103]. Since kappa shows substantial agreement, the remaining software artifacts categorization was conducted by only 1 coder. Our manual analysis led to the creation of a taxonomy of documentation related artifacts with 7 distinct categories. A description of each category follows:

1. **Contributors’ Guide** contain information targeting the contributors to the project such as how to begin contributing to the project, the review process, tips on debugging, etc.
2. **Design Documents** contain information about the design of the project, such as design patterns and design decisions, underlying project framework and architecture, as well as version compatibility details.
3. **License** contain information about copyright and the type of licenses the project operates under.
4. **List of Contributors** contain information about and credit to the authors and maintainers of the project, including author names, their roles, and contact information.
5. **Release Notes** are usually documents shared with end users or clients and outline specific version changes, bug fixes, or enhancements made to the project.
6. **Requirement Documents** often contain functional and non-functional requirements, use cases, and other software specifications that target expected user interactions.
7. **Setup Files** contain all artifacts that have to do with project setup. Examples include manifest files, make files, configuration files, and version requirement files.

During the manual classification, we identify 342 unique features that characterize the categories in the above taxonomy. Some of those are based on their frequency of occurrence in artifacts, while others are identified by the coders. We observe that five features are



Table 4.13: Extension distribution in the sampled projects.

<b>File Extension</b>	<b>Num. of Projects</b>	<b>%</b>	<b>Num. of Files</b>	<b>%</b>	<b>Cum. %</b>
no extension	383	100.00%	13,706	5.85%	5.85%
md	262	68.41%	1,853	0.79%	6.64%
html	162	42.30%	3,203	1.37%	8.01%
txt	162	42.30%	7,828	3.34%	11.35%
png	153	39.95%	8,450	3.61%	14.96%
js	152	39.69%	9,921	4.23%	19.19%
css	132	34.46%	1,405	0.60%	19.79%
xml	115	30.03%	6,147	2.62%	22.41%
json	109	28.46%	1,542	0.66%	23.07%
jpg	97	25.33%	1,300	0.55%	23.63%
java	79	20.63%	3,582	1.53%	25.15%
ico	65	16.97%	96	0.04%	25.20%
svg	59	15.40%	435	0.19%	25.38%
sh	58	15.14%	1,265	0.54%	25.92%
gif	56	14.62%	2,614	1.12%	27.04%
properties	53	13.84%	164	0.07%	27.11%
py	53	13.84%	15,147	6.46%	33.57%
h	49	12.79%	48,448	20.68%	54.25%
php	42	10.97%	2,645	1.13%	55.38%
jar	42	10.97%	260	0.11%	55.49%
ttf	42	10.97%	131	0.06%	55.55%
yml	39	10.18%	264	0.11%	55.66%
woff	37	9.66%	79	0.03%	55.69%
eot	36	9.40%	78	0.03%	55.73%
pdf	35	9.14%	400	0.17%	55.90%
rb	32	8.36%	2,267	0.97%	56.86%
scss	29	7.57%	780	0.33%	57.20%
c	28	7.31%	43,056	18.38%	75.57%
sln	28	7.31%	100	0.04%	75.62%
lock	27	7.05%	41	0.02%	75.63%
conf	26	6.79%	270	0.12%	75.75%
bat	26	6.79%	51	0.02%	75.77%
plist	25	6.53%	266	0.11%	75.88%
cpp	25	6.53%	1,581	0.67%	76.56%
cache	23	6.01%	59	0.03%	76.58%
log	22	5.74%	133	0.06%	76.64%
config	21	5.48%	77	0.03%	76.67%
map	20	5.22%	123	0.05%	76.73%

Table 4.14: Sample list of features.

Document Type	#	Example Features
Contributors' Guide	26	contribute, welcome, checkout, severity
Design Document	10	architecture, design, framework, layer
License	30	disclaimer, free, law, reproduction
List of Contributors	18	authors, instructions, maintainers, thank
Release Notes	30	added, bug, date, fixed, improve, version
Requirement Document	10	feature, functionality, support, requirement
Setup Files	25	build, configure, defaults, ignore, manifest

not present in any of the files in our oracle. We remove those features and retain the remaining 337 features that we will use for the automatic artifact classification. Table 4.14 shows examples of the features we used to identify each category and the distribution of artifacts in our oracle. The complete list of features can be found in our online replication package. Based on the in-depth analysis and manual classification of 894 artifacts, the following conclusion was drawn:

**RQ<sub>4.1</sub>** Summary: Some software artifacts can be categorized solely using heuristics based on file names and extensions. However, other artifacts that are documentation related require deeper analysis and identification of characterizing features to be classified.

#### 4.4.2 RQ<sub>4.2</sub>: How accurate is the proposed approach for automatic software artifact classification?

In this section we evaluate the performance of the automatic artifact classification. We do not evaluate the classification of non-document related artifacts, i.e., those listed in Table 4.12 as those are correct by construction. Table 4.15 contains the results of applying ML algorithms using 10-fold cross-validation. Results per class as well as the micro and macro averages across classes are reported. Overall, Naïve Bayes Multinomial has the best performance with a micro average precision of 0.80, 0.76 recall, 0.76 F-measure, 0.73 MCC, and 0.95 ROC. The high values for MCC and ROC indicate that the classifier performs very well on the validation dataset.

Values in bold indicate the best performance achieved per class for both precision and recall. For example, at 0.74, J48 is able to achieve the highest precision for the class Release Notes relative to the other classifiers. However, at 0.83, Naïve Bayes Multinomial achieves the highest recall for Release Notes. Each algorithm achieves the best precision and recall performance for at least one class, therefore, different algorithms may be better suited to classify instances from different classes. Using ensemble techniques, such as voting we are able to combine the predictive power of several algorithms that perform well on unique classes, to create one classifier with improved performance across all classes.

Table 4.16 contains the results of classifiers used in Table 4.15 combined using ensemble learning. Specifically, the classifiers are combined using majority vote. Results in Table 4.15 indicate that Naïve Bayes Multinomial performs the best on several different classes, therefore

Table 4.15: Performance of individual classifiers and 10-fold cross-validation on the training dataset.

Classifier	Parameters	Class	Precision	Recall	F-Measure	MCC	ROC
Naïve Bayes Multinomial	Default	Requirement Document	0.35	<b>0.70</b>	0.47	0.45	0.92
		Design Document	0.63	0.50	0.56	0.53	0.93
		Release Notes	0.69	<b>0.83</b>	0.76	0.69	0.97
		Setup Files	<b>0.86</b>	0.48	0.62	0.59	0.94
		License	<b>0.94</b>	<b>1.00</b>	0.97	0.96	1.00
		List of Contributors	<b>0.89</b>	<b>0.89</b>	0.89	0.87	0.99
		Contributors' Guide	<b>0.86</b>	<b>0.69</b>	0.77	0.73	0.89
		Micro Average	0.80	0.76	0.76	0.73	0.95
		Macro Average	0.74	0.73	0.72	0.69	0.95
		SMO Poly Kernel	Default	Requirement Document	<b>0.40</b>	0.40	0.40
Design Document	0.43			0.30	0.35	0.66	0.91
Release Notes	0.70			0.77	0.73	0.66	0.90
Setup Files	0.77			<b>0.92</b>	0.84	0.81	0.96
License	<b>0.94</b>			0.97	0.95	0.94	0.99
List of Contributors	0.82			0.78	0.80	0.77	0.95
Contributors' Guide	0.81			0.65	0.72	0.68	0.87
Micro Average	0.75			0.76	0.75	0.71	0.93
Macro Average	0.69			0.68	0.69	0.65	0.92
Random Forest	#Trees 500			Requirement Document	<b>0.40</b>	0.20	0.27
		Design Document	<b>1.00</b>	0.30	0.46	0.53	0.97
		Release Notes	0.65	0.73	0.69	0.60	0.94
		Setup Files	0.71	0.88	0.79	0.74	0.95
		License	0.91	1.00	0.95	0.94	1.00
		List of Contributors	0.87	0.72	0.79	0.77	0.98
		Contributors' Guide	0.64	<b>0.69</b>	0.67	0.59	0.93
		Micro Average	0.74	0.74	0.72	0.69	0.96
		Macro Average	0.74	0.00	0.69	0.63	0.95
		J48	MinNumObj 4	Requirement Document	0.27	0.30	0.29
Design Document	0.67			<b>0.60</b>	0.63	0.61	0.84
Release Notes	<b>0.74</b>			0.67	0.70	0.63	0.87
Setup Files	0.45			0.52	0.48	0.37	0.76
License	0.93			0.93	0.93	0.92	0.98
List of Contributors	0.43			0.50	0.46	0.38	0.79
Contributors' Guide	0.55			0.46	0.50	0.41	0.82
Micro Average	0.62			0.61	0.62	0.55	0.84
Macro Average	0.58			0.57	0.57	0.51	0.82

Table 4.16: Performance of the classifiers using ensemble learning and 10-fold cross-validation on the training dataset.

Classifier	Class	Precision	Recall	F-Measure	MCC	ROC
Majority Vote	Release Notes	0.85	0.84	0.84	0.81	0.90
(2*Naïve Bayes Multinomial, SMO Poly Kernel, J48, and Random Forest)	Contributors' Guide	0.90	0.78	0.84	0.81	0.88
	List of Contributors	0.99	0.86	0.92	0.91	0.93
	Design Document	0.74	0.51	0.60	0.59	0.75
	License	0.98	1.00	0.99	0.99	1.00
	Requirement Document	0.39	0.66	0.49	0.46	0.79
	Setup Files	0.74	0.79	0.77	0.72	0.87
	Micro Average	0.85	0.82	0.83	0.80	0.90
	Macro Average	0.80	0.78	0.78	0.76	0.87

Table 4.17: Performance of the classifiers using ensemble learning and 10-fold cross-validation on the testing dataset.

Classifier	Class	Precision	Recall	F-Measure	MCC	ROC
Majority Vote (2*Naïve Bayes Multinomial, SMO Poly Kernel, J48, and Random Forest)	Release Notes	0.50	0.80	0.62	0.54	0.82
	Contributors' Guide	0.90	0.82	0.86	0.83	0.90
	List of Contributors	0.86	1.00	0.92	0.92	0.99
	Design Document	1.00	0.40	0.57	0.62	0.70
	License	1.00	0.90	0.95	0.94	0.95
	Requirement Document	0.33	0.14	0.20	0.15	0.55
	Setup Files	0.75	0.90	0.82	0.78	0.92
	Micro Average	0.76	0.75	0.73	0.70	0.85
	Macro Average	0.76	0.71	0.70	0.68	0.83

we increase the weight of its vote during classification by two to create a weighted majority vote, which has shown to be effective in similar text classification research [152]. As compared to the best performing single classifier, majority vote yields a micro average precision of 0.85, which is a 5% increase, recall increases by 6% to 0.82, F-Measure increases by 7% to 0.83. MCC increases by 7% to 0.80 and ROC decreases to 0.90, which still indicates strong performance.

Requirement Document is the class with the lowest performance using both single classifiers and voting. However, using voting we are able to achieve a better balance between precision and recall. The best precision and recall for the class are both at 0.40 for single classifiers, however, with ensemble learning precision drops by only 0.01 and recall increases by 0.26. Overall, voting improves the performance in terms of precision and recall across all classes. The only exception is with the class Setup Files for which SMO Polynomial Kernel is able to achieve a 3% higher precision and 13% higher recall. Despite this, comparing the micro average for all classes of SMO Polynomial Kernel to the ensemble approach, the performance trade off is a 10% increase in precision, 6% increase in recall in favor of the ensemble approach.

In order to evaluate the model generated by the majority vote algorithm, we run the classifier on a newly generated oracle, the testing dataset, and analyze the results. Table 4.17 contains the results of the classifier on the second oracle of 59 data points. Overall, results for classes Contributors Guide, List of Contributors, Design Documents, License, and Setup Files are very similar, in term of F-Measure, MCC and ROC, to the performance obtained on the first oracle. Release Notes and Requirement Documents are two categories that perform significantly worse with 0.35 decrease in precision for Release Notes and 0.52 decrease in recall for Requirement Documents. The results for these two classes affect the overall micro and macro averages. 3 out of 7 instances from the Requirement Document class are categorized as Release Notes and 2 out of 10 instances of Release notes are categorized as Requirement Documents. We investigate the ML features across the different types of artifacts to understand the drop of performance in the testing dataset. Our analysis leads to two observations. First, we note that there is a significant decrease in the number of documents containing the features for Requirement Documents in the testing dataset. The second observation is that there is an increased overlap of features between Requirement Documents and Release Notes in the testing dataset. One explanation could be due to the fact that the features we manually created are not representative of Requirement Documents.

Another explanation could be due to fact that Requirement Documents in the second oracle are considerably smaller in size compared to the Requirement Documents in the first oracle. Thus, there might not be enough textual content, i.e., features, in the second oracle for the ML algorithms to perform well. We plan to further investigate and try to improve the performance of ML features regarding the Release Notes and Requirement Document artifacts in our future work by adding more documents to the training set and by comparing the performance of manually extracted features to that of automatically extracted features using information retrieval approaches.

**RQ<sub>4.2</sub>** Summary: Combining different ML algorithms through ensemble learning, we are able to automatically classify documentation related software artifacts with an average precision of 85% and recall of 82% using 10-fold cross-validation on the validation dataset—an oracle of 149 data points. Using the same classifier on a testing dataset of 59 new data points, our approach achieves an average precision of 76% and a recall of 75%.

## 4.5 RQ5: What types of artifacts are created during open-source software development?

To explore the types of artifacts created during open-source software development, we run our classification approach on the entire sample set of 383 projects. Table 4.18 contains the predicted distributions of various documentation and non-documentation related artifacts created during open-source project development. “Num. of Projects” (%) reports on the number (percentage) of sampled applications that contain each type of artifact. Overall, the most common type of artifacts are source code, setup, miscellaneous, and archive, which are identified in over 50% of the applications. The least common type of artifacts are disk image and audio, which are identified in less than 5% of the applications.

“Num. of Files” (%) in Table 4.18 reports on the number (percentage) of artifacts from each category across all sampled applications. There is a total of 87,619 software artifacts in the sample applications. We observe that documentation related artifacts make up only 6.12% of all files. Further more, design documents and requirement documents only make up 0.42% and 0.68% respectively. Setup files account for 3.57% of the total artifacts. As expected, source code makes up 56.79% of the entire artifacts collection.

Focusing on documentation, we observe that 5.74% and 10.18% of the projects contain design and requirement documents, respectively. Taking into consideration that 4 projects contain both design and requirement documents, the combination of projects that contain either type makes up 14.88% of the sampled applications ( $22+39-4=57$ ). Although documentation related artifacts only accounts for a small portion of the available artifacts, open-source projects can still be a good resource for researchers for such artifacts.

**RQ<sub>5</sub>** Summary: Using our automatic artifact classification approach, we confirm that open-source projects provides a variety of software artifacts. Approximately 14.88% of the projects contain either design or requirement documents.

Table 4.18: Distribution of the different types of software artifacts in the sampled projects.

<b>Software Artifacts</b>	<b>Category</b>	<b>Num. of Projects</b>	<b>%</b>	<b>Num. of Files</b>	<b>%</b>
Documentation	Design Documents	22	5.74%	371	0.42%
	List of Contributors	33	8.62%	134	0.15%
	Requirement Documents	39	10.18%	592	0.68%
	Contributors' Guide	54	14.10%	389	0.44%
	License	84	21.93%	259	0.30%
	Release Notes	93	24.28%	489	0.56%
	Setup Files	235	61.36%	3,130	3.57%
Subtotal				5,364	6.12%
Non-Documentation	Disk Image	1	0.26%	4,209	4.80%
	Audio	5	1.31%	83	0.09%
	Project	25	6.53%	68	0.08%
	Font	31	8.09%	201	0.23%
	Application	32	8.36%	121	0.14%
	Testing Code	92	24.02%	3,766	4.30%
	Image	126	32.90%	10,212	11.66%
	Source Code	217	56.66%	49,680	56.70%
	Misc	236	61.62%	13,380	15.27%
Archive	236	61.62%	535	0.61%	
Subtotal				82,255	93.88%
Total				87,619	100%

## 4.6 RQ6: How do experts assess the quality of traceability datasets?

Figure 4.11 shows the demographic information about the participants of the survey. Out of the 23 experts, 14 are from within the US, 5 from Europe, 2 from Asia, and 2 from Canada. Twenty-two experts are researchers from academia and one is from industry. In the following, we present the answers for each of these questions:

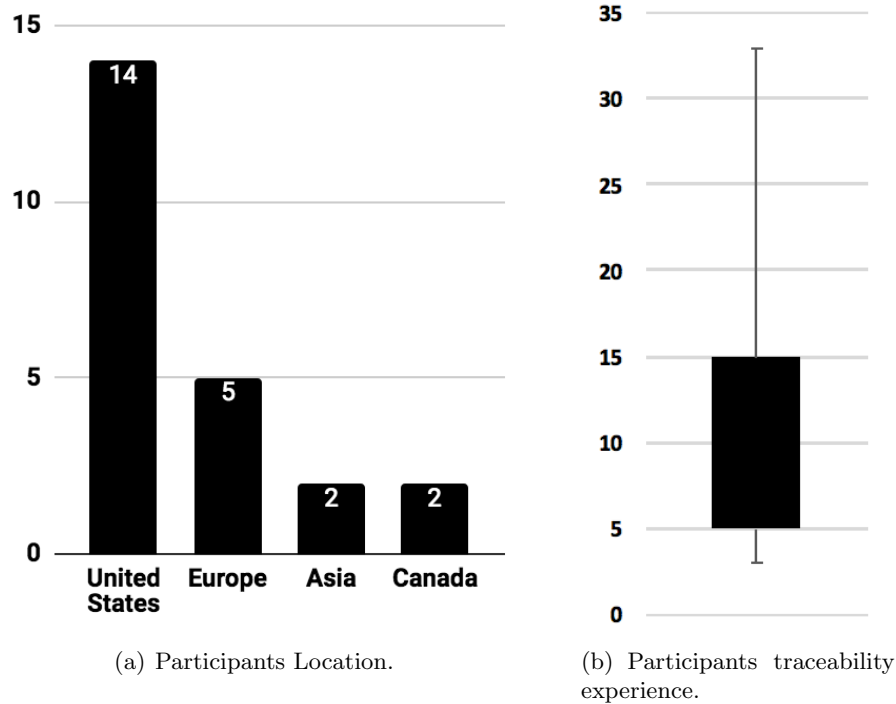


Figure 4.11: Demographic information about the participants.

### 4.6.1 RQ6.1: What are the quality attributes that researchers are looking for when they select datasets?

The descriptive answers provided by the experts participated in the survey were analyzed using open coding practices. First, we highlight a descriptive tag for each answer, then we created a code for these descriptions and the codes were converted into categories defining quality attributes of the datasets.

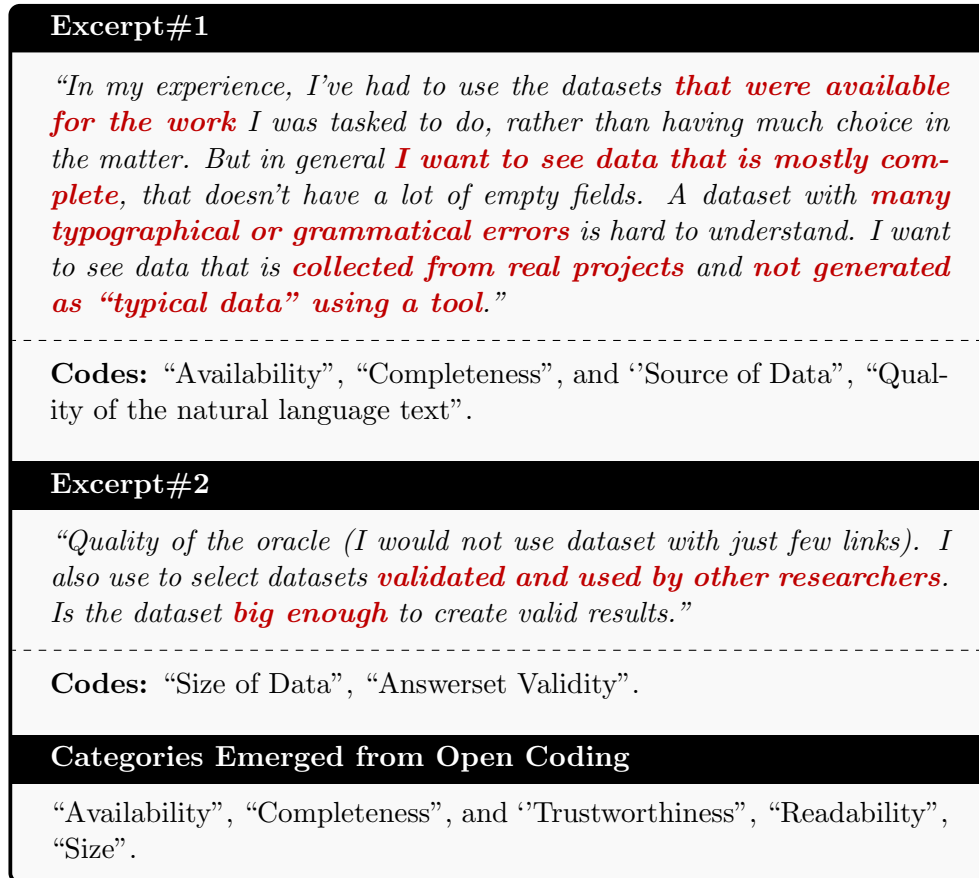


Figure 4.12: Examples of Excerpt from online survey responses, identified codes in open coding activities, and emerged categories.



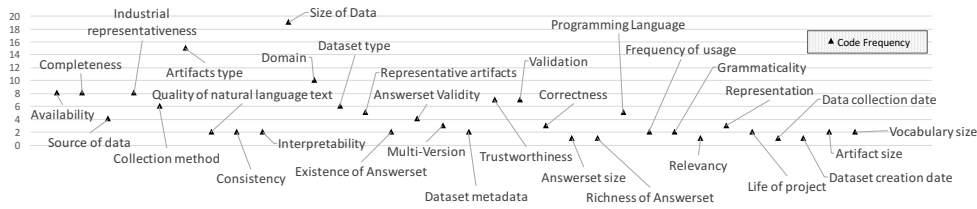


Figure 4.13: Codes emerged from open coding activities and their frequencies.

Figure 4.12 shows excerpts from answers provided by two different experts, and descriptive tags identified as bold font and red color. The codes generated by the authors are shown below each excerpt and categories emerged from this coding activity are shown in the last part of this figure.

For instance, in case of the examples shown here, the following codes were discovered from the responses: “Availability”, “Completeness”, “Source of Data”, “Quality of natural language text”, “Size of Data”, and “AnswerSet Validity”. This process was repeated for all the answers provided by the participants. The identified codes from the entire dataset were discussed by the research team and then grouped into logical categories. Example of emergent categories are “Availability”, “Completeness”, and “Developer”, “Readability”, “Size”, and “Trustworthiness”.

Figure 4.13 shows all the codes emerged through open coding of the survey results and their frequency. These codes were used to summarize the respondent’s opinions about what attributes do they consider when reasoning about datasets quality. These are intermediary data points, in the next step of the grounded theory, these codes were grouped into categories until no further categories could be discovered.

Figure 4.14 shows all the emerged categories (quality attributes) and their frequency. Relevancy of the datasets to research problem, their size and trustworthiness are top three most common quality attributes discussed by the researchers. Each of these 11 categories, their descriptions, and the codes associated with them are summarized below:

- Relevancy:** Refers to the provision of information which is in accordance with the task at hand. **Artifacts type:** Representing various forms of data generated from software engineering activities. Examples of artifact types are requirements document, test cases, design documents, test plans, assurance cases and etc. This code is observed in the responses of 15 participants. Several participants highlighted the importance of having heterogeneous artifacts. One expert reports that: “*Type of the artifacts in a dataset is useful to measure the generalizability of the conclusions. I may think of (and experienced with) techniques that work for some software artifacts (e.g., requirements) but that are not applicable for other artifacts (e.g., test cases).*”

**Representativeness:** whether the dataset or artifacts are representative of a population. This code was observed in the responses of 5 participants. One expert points out that representativeness is important for example when needing to demonstrate that the results of an “*approach are applicable to an industry-wide problem*”. Another expert mentions that she will use “*industrial or industry-like datasets*”.

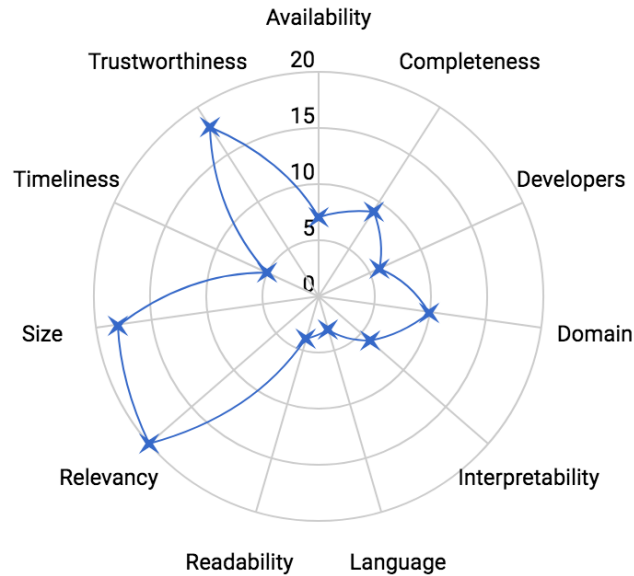


Figure 4.14: Quality attributes used by the experts to select datasets and their frequencies in the experts' responses.

- **Size:** Reflecting how big is the dataset, and how many artifacts are for each artifact type.

Dataset size: This code was observed in the responses of 19 participants. One participant reports that a lot of his work uses machine learning approaches and thus the number of data points, i.e., the size of the dataset really matters. Two other participant highlight that size is important to generate “*valid research results*”.

Artifact size: reflects how big is an artifact in terms of the number of entities (requirements, methods, classes, etc.) within that artifact. This code was observed in the responses of 2 participants. One expert reports that it is important to have many artifact types and many artifacts per type.

Size of the vocabulary: Reflecting how big is a textual artifact and how diverse is its vocabulary. This code was observed in the responses of one participant.

Representative artifacts: To what extent the provided artifacts are correctly representing the required artifacts type. This code was observed in the responses of 5 participants. For instance, one participant points out that “feature requests in Jira are a poor proxy for requirements”.

- **Trustworthiness:** Is the information provided in the dataset accepted to be correct, true, real, and credible.

Answerset trustworthiness: Are the trace links provided in the answerset accepted to be correct and credible. This code was observed in the responses of 4 participants. One participant highlights that for him it is important to have a “full understanding how this data is obtained”. Another participant mentions that “*if non-project members created links after the fact, the dataset cannot be considered realistic*”.

Validation: Is the dataset been vetted and validated previously by researchers. This code was observed in the responses of 7 participants. One participant reports that “*Accuracy above all. False positives will let me draw to completely biased conclusions.*”

Frequency of Usage: How frequently is the dataset used by the other researchers. This code was observed in the responses of 2 participants.

Generation approach: indicates whether the dataset is established and vetted manually or whether it is generated automatically by a tool. This code was observed in the responses of 2 participants. One participant reports that “*A dataset is trustworthy to me when I know how it has been generated and in general the generation process is replicable.*”

- **Language**: Are the provided artifacts written in different languages (natural languages, programming languages).

Natural languages: This code was observed in the responses of 2 participants.

Programming languages: This code was observed in the responses of 5 participants.

- **Completeness**: Refers to the degree to which all required information is present in a particular dataset. The following codes have resulted the creation of this category:

Presence of Source-Target artifacts: at least 2 types of artifacts are present in the dataset. This code was observed in the responses of 5 participants. One expert adds that “*Completeness - that is, I want to have more than just the traceability matrix, for example, including the source code and all relevant artifacts.*”

Answerset: Is there an answerset (Oracle) that was provided with the dataset. This code was observed in the responses of 2 participants.

- **Availability**: Is the extent to which the dataset is present, obtainable and ready for use. This code was observed in the responses of 7 participants.
- **Interpretability**: It refers to whether the information is represented using an appropriate notation and whether the machine is able to process the data. This code was observed in the responses of 6 participants.
- **Developers**: Team responsible for creating the dataset or answerset. The following codes have resulted the creation of this category:
  - Source of Dataset: Open source community, industrial, or academic. This code was observed in the responses of 4 participants.
  - Source of Answerset: Professionals, students, or tools. This code was observed in the responses of 2 participants.
- **Readability**: Refers to correctness and appropriateness of the artifact in its underlying language.

Grammaticality: Any natural language artifacts needs to be proper and free of grammar issues. This code was observed in the responses of 2 participants.

Consistency: whether consistent terminology, style, and the argument are present in the data. This code was observed in the responses of 2 participants.

- **Timeliness**: Refers to the date of the dataset or answerset creation. The following codes have resulted the creation of this category:

Multi-Version: Refers to multiple version of the artifacts or the source code that was provided within the dataset. This code was observed in the responses of 3 participants. One expert mentions that “*There should be enough information about the relationships between the source and target artifacts so that I can evaluate the traceability. For example, the information can be multiple versions of the project. The co-evolution of the requirements and code can be obtained. These are proof of the trace links.*” Furthermore, the same expert emphasizes the timeliness of projects, “*I like the active and hot projects, because more people participating in the project, higher probability that I can get more related and diverse data.*”

Data collection date: Refers to the date in which the dataset was obtained. This code is observed in the responses of one participant.

Data creation date: Refers to the date in which the dataset was obtained. This code is observed in the responses of one participant.

- **Domain**: A specified sphere of activity or knowledge having a common set of requirements, terminology, and functionality. This code is observed in the responses of 10 participants. Participants have different opinions whether the quality attributes used to assess the dataset change for different domains.

#### 4.6.2 RQ6.2: What dataset qualities have an impact on the meaningful conclusions being drawn from a research project?

Based on the feedback collected from participants, dataset size is the most important quality attribute impacting the conclusion drawn from research in this area. After this attribute, the trustworthiness of the dataset and relevancy of the datasets to the domain and tasks at hand are the next two most important attributes. Figure 4.15 illustrates all attributes identified by the experts and the number of experts who mention that attribute.

#### 4.6.3 RQ6.3: What are the datasets quality-attributes that could impact the generalizability of research results?

Similarly, we obtained experts’ opinion on the topic of generalizability of the research results and its association with the dataset quality. Figure 4.16 illustrates all the attributes identified by the experts and the number of experts who have mentioned that attribute. Relevancy and size are the most frequently reported quality attributes followed by trustworthiness and domain.

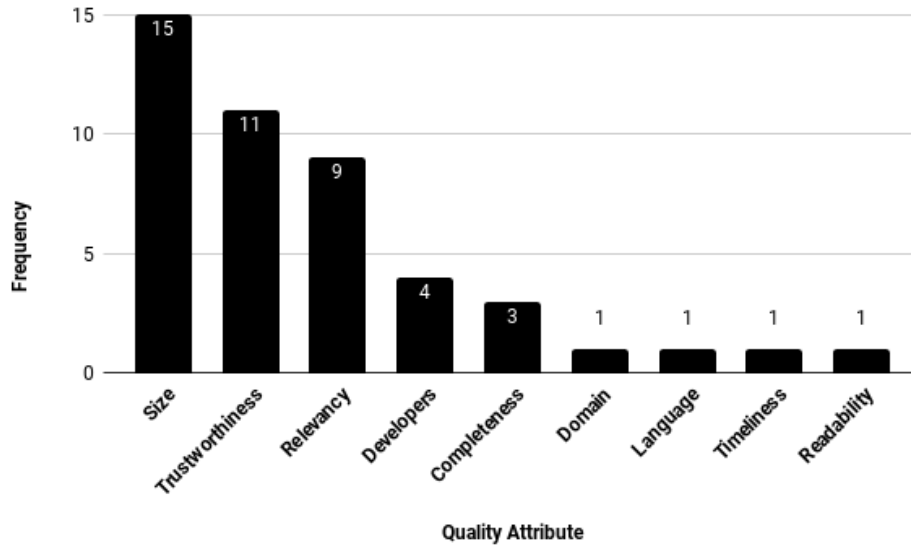


Figure 4.15: Expert’s opinion on the relationship between the dataset qualities and research conclusions.

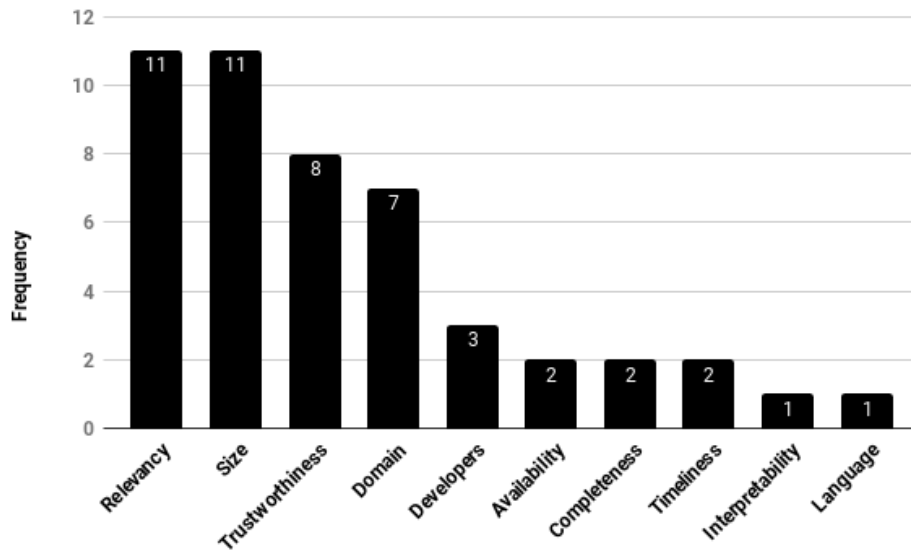


Figure 4.16: Expert’s opinion on the relationship between the dataset qualities and generalizability of the results.

**RQ<sub>6</sub> Summary of findings:** Researchers use a combination of eleven attributes to reason about the quality of scientific datasets and select the appropriate ones. Among these attributes, datasets size, trustworthiness, and relevancy of datasets to research task at hand are three important factors impacting the validity of research conclusions and their generalizability.

## 4.7 RQ7: Does the existing framework for evaluating the quality of traceability datasets captures the relevant characteristics that experts are looking for?

Subject matter experts were presented with the T-DQA quality sub-dimensions and were asked to rank them on a five-point Likert scale, one representing not important and four to capture the importance of each quality attribute. They were also given an option of selecting “Do not have an opinion”. This option helps minimize the risks in cases that the subjects are uncertain about their answers. Figure 4.17 summarizes the expert’s opinion about the importance of T-DQA sub-dimensions.

**Dataset size:** 65.2% of the surveyed experts consider the dataset size to be a very important attribute to assess dataset quality in the domain of requirements traceability. That supports the inclusion of this characteristic in T-DQA framework.

**Dataset domain:** Eleven of the surveyed experts (47.8%) consider that the dataset domain is important or very important quality attribute. Eight participants (34.8%) report the quality attribute as moderately important. One participant shares: *“if it is a safety critical system then completeness and consistency is even more important. Accuracy is also important.”* However, other participants report that the domain should not be playing a role: *“Ideally you want all datasets to be of the same quality, regardless of the application.”*

**Artifact types:** Seven experts (30.4%) consider artifact types very important, and ten experts (43.5%) consider artifact types important quality attribute of the datasets. Experts also highlight the importance of artifacts granularity.

**Availability:** Sixteen experts (69.6%) consider the availability of dataset as a very important attribute for the datasets quality. Six experts consider it important. This reflects the importance of this attribute in desirability of a dataset for researchers.

**Completeness of the dataset artifacts:** Eleven experts (47.8%) consider completeness of the dataset artifacts as a very important attribute for the datasets quality. Eight experts (34.8%) consider it as important.

**Representation:** Nine experts (39.1%) consider representation to be an important or very important attribute. Seven experts (30.4%) report that this attribute is moderately important for assessing the quality of the dataset.

**Dataset Developer:** For this quality attribute, eleven experts (47.8%) consider it to be important or very important quality attribute. Seven experts (30.4%) consider dataset developer as a moderately important quality attribute. There have been conflicting responses when it comes to this attributes. One expert reports that *“researchers should primarily focus on industry datasets.”*, there are other experts that agree with this opinion. Another expert

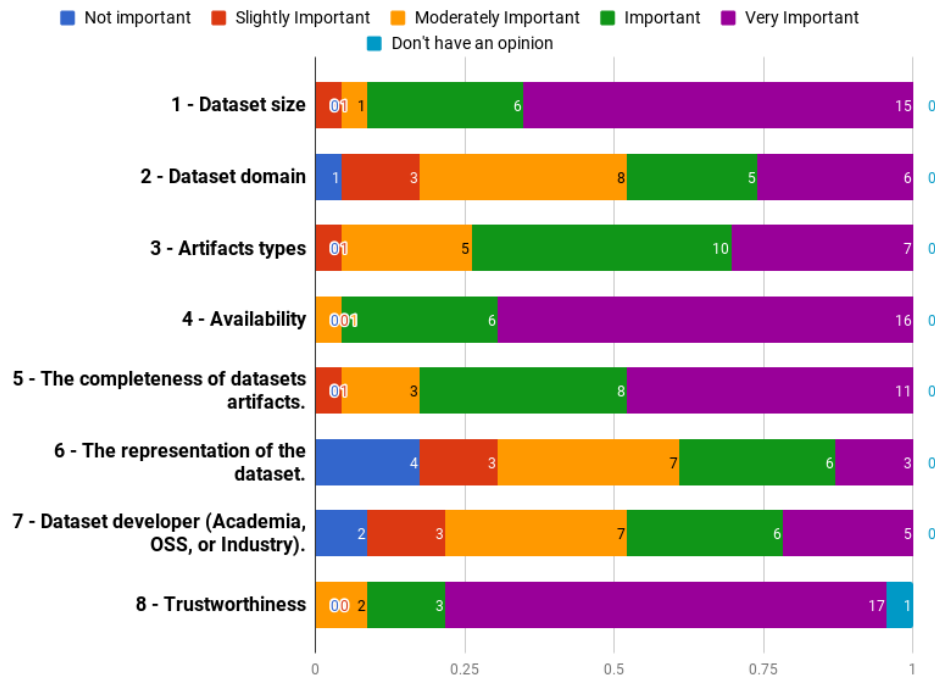


Figure 4.17: Importance of T-DQA attributes from participant’s perspective.

mentions that “Many OSS projects are higher quality than closed-sourced ones I have encountered.” Another expert argues that “it depends what the academic project looks like (toy ones like EasyClinic are too small). iTrust has been used a lot even though it is academic—but its a poor substitute. OSS definitely \*not\* good enough—unless your interest lies specifically in building tools for OSS—in which case those datasets are clearly perfect.”

There are other experts that argue that this quality attribute is dependent on the context of research and other attributes of the dataset itself. For instance, one researcher mentions that “There is no definite answer for Dataset Developer. It depends. They can be good enough, that it must be assessed.” Another expert reports that “One of the fundamental aspects of any dataset is the incentive for the engineers involved in creating and maintaining it. If an OSS project carefully maintained traceability to support regression testing, that would be very interesting. If an academic project maintained traceability because the developers considered it valuable (not just students trying to get good grades...) for long-term evolution of a product—yes, very interesting as well.”

**Trustworthiness:** Seventeen experts (73.9%) consider the trustworthiness of dataset as a very important attribute to assess dataset quality in the domain of requirements traceability. This is the quality attribute with the highest number of participants considering it very important.

**RQ7 Summary of findings:** Traceability experts identify the quality attributes defined in T-DQA important, in particular Trustworthiness (73.9%), Availability (69.6%), and Dataset Size (65.2%).

### 4.7.1 Discussion

In earlier sections (4.6 and 4.7), we used a grounded theory-based approach to answer two research questions of **RQ6** and **RQ7**

Such approach can be particularly useful for researchers whose topic of interest has not been subject to systematic analysis before and theories in the domain are incomplete or inexistent. Since the topic of dataset quality in the domain of requirements traceability has not been analyzed and articulated in-depth, we considered that grounded theory would help us in deriving a theory around. This analysis helps better understand how researcher reason about the quality of their scientific datasets and how do they choose among various datasets.

Our analysis showed that while T-DQA represents a set of quality attributes that relevant to the surveyed experts, it does not capture all the attributes that they use when reasoning about dataset quality. We have used the results obtained in **RQ6** and **RQ7** to extend the T-DQA framework.

Table 4.19 shows the updated version of this framework (i.e., *T-DQA v.2*) obtained based on the feedback collected from 23 software traceability experts. The light gray color shows the quality attributes modified and the dark gray color shows new quality attributes that are added in T-DQA v.2.

Three new quality attributes are added to the framework: *Timeliness* (suggested by 5 experts), *Size* (suggested by 18 experts) and *Readability* (suggested by 5 experts). Furthermore, the *Language* sub-dimension was modified to cover both programming languages and different natural languages.

We did not remove any attribute from T-DQA, as the majority of the surveyed experts consider all quality attributes defined by the T-DQA framework from moderately to very



important.

Participants report that since requirements traceability research significantly relies on textual artifacts beside source code or models, metrics such as “Size of vocabulary” are necessary to reason about the appropriateness of a dataset for research. Furthermore, consistency of natural language, and statements as well as correctness in terms of language grammar, are important indicators of high-quality datasets.

Trustworthiness of scientific datasets can also be reasoned in terms of collection approach, validation and vetting procedure, and correctness of data or answersets. One traceability expert reports that two major challenges in obtaining high quality dataset are the correctness and completeness of the answerset. Often traceability datasets lack answerset, may have an answerset with false positive links or missing links (incomplete). Such metrics can be used to reason about the quality and appropriateness of a dataset within the context of researchers projects. Another participant reports that a dataset is trustworthy when “*I can verify it myself and when I know who produced it.*”

Relevancy of datasets to researchers problems at hand is a contextual quality metric. One metric discovered through this study was representativeness of datasets or artifacts. Experts consider a dataset high quality if it represents the industrial domain of their interest or the type of artifact they are looking for.

Table 4.19: Updated Framework: *T-DQA v.2*.

Dimension	Sub-dimension	Definition	Metrics	Metric type
Accessibility	Availability	Availability of a dataset is the extent to which data is present, obtainable and ready for use.	Dataset can be downloaded from the link provided.	Binary
	Licensing	Licensing is defined as the granting of permission for a consumer to re-use a dataset under defined conditions.	license agreement exists.	Binary
	Storage	Where is the Dataset Stored?	Private Server/ Organizational Server /Public	Categorical
Intrinsic	Domain	A specified sphere of activity or knowledge having common set of requirements, terminology, and functionality	Application Domain	Categorical
	Completeness	Completeness refers to the degree to which all required information is present in a particular dataset.	Source-Target artifacts are present. (at least 2 types of artifacts are present in dataset)	Binary
			Answerset completeness Answer Set is Present	Binary
	Dataset Developer	Team responsible for creating the dataset or answerset	Dataset: Open source community, industrial, or academic Answerset: Professionals, students, or tools	Categorical
	Language	Describes the natural language or the programming language used to write software artifacts.	Java/ C++ etc. Natural Language	Categorical
	Timeliness	Refers to the date of the dataset or answerset creation as well as the period of time its been in use.	Active	Binary
			Popular Multi-version Collection Date, Creation Date	Binary Binary Date
Size	Reflecting how big is the dataset, and how many artifacts are for each artifact type.	Dataset size, Answerset size, Software artifact size, Size of the vocabulary	Numerical	
Contextual	Relevancy	Relevancy refers to the provision of information which is in accordance with the task at hand and important to the users' query.	Type of artifacts (Req., UML Diagrams, Code, Test, etc.)	(Type, Numerical)
			Industrial/ Domain Representativeness Artifact Representativeness	Binary Binary
	Trustworthiness	Trustworthiness is defined as the degree to which the information is accepted to be correct, true, real, and credible.	Dataset Source	Categorical
			Frequency of Usage Manual Collection Method	Numerical Binary
			Automatic Generation by Tools Dataset Correctness, Answerset Correctness	Categorical Binary
Representational	Interpretability	It refers to technical aspects of the data, that is, whether information is represented using an appropriate notation and whether the machine is able to process the data.	detecting the use of appropriate language, symbols, units, datatypes and clear definitions. Structure of the data	Categorical
	Readability	Refers to correctness and appropriateness of the artifact in its underlying language.	Grammaticality Consistency	Binary Binary

## 4.8 T-DQA Web-tool support

As we mentioned earlier that T-DQA web-tool is an implementation of our previously proposed and updated quality framework (T-DQA v2). The main goal of this tool is to provide traceability researchers with a mean to easily find high-quality datasets that meet their needs. Figure 4.18 shows the main-page of the web-tool. The left panel holds all the quality metrics that were listed in Table 3.10 in the form of filters that researchers can select from. On the right panel, the list of all the 37 datasets. For each dataset as shown in Figure 4.19, the following details are represented:

1. Dataset Name
2. Dataset Domain
3. Dataset Size (the total number of artifacts)
4. Dataset Artifacts Format
5. Dataset Description
6. More Details
7. Download Dataset in a ZIP file

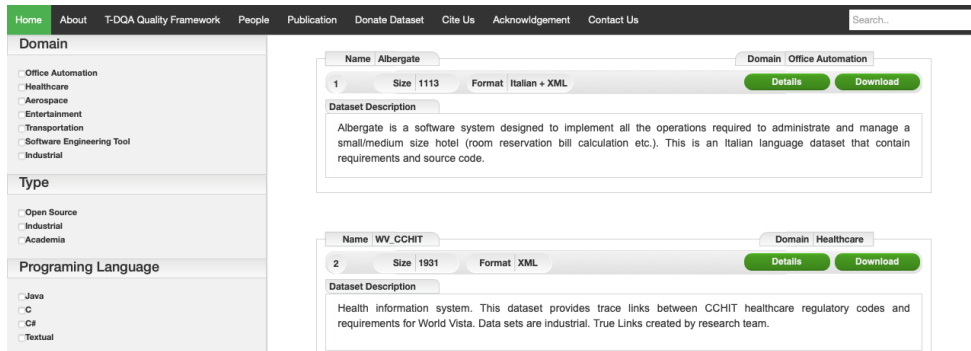


Figure 4.18: T-DQA Web-Tool Interface

Figure 4.20, Shows an example of T-DQA web-tool usage by researcher who is looking for a dataset that has the following quality characteristics:

- Domain: Office Automation
- Type: Industrial
- Programming Language: Java

- Popular: Yes
- Multi Versions: No
- Artifacts Type: Requirements and Source Code

As shown in the figure above, the dataset that matches the quality filters selection is appearing on the right panel where the researcher can choose either to go over its details by clicking on the details button or download the dataset by clicking on the download button.

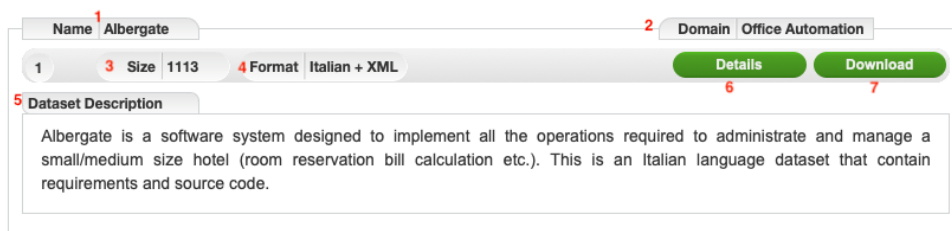


Figure 4.19: Datasets Details

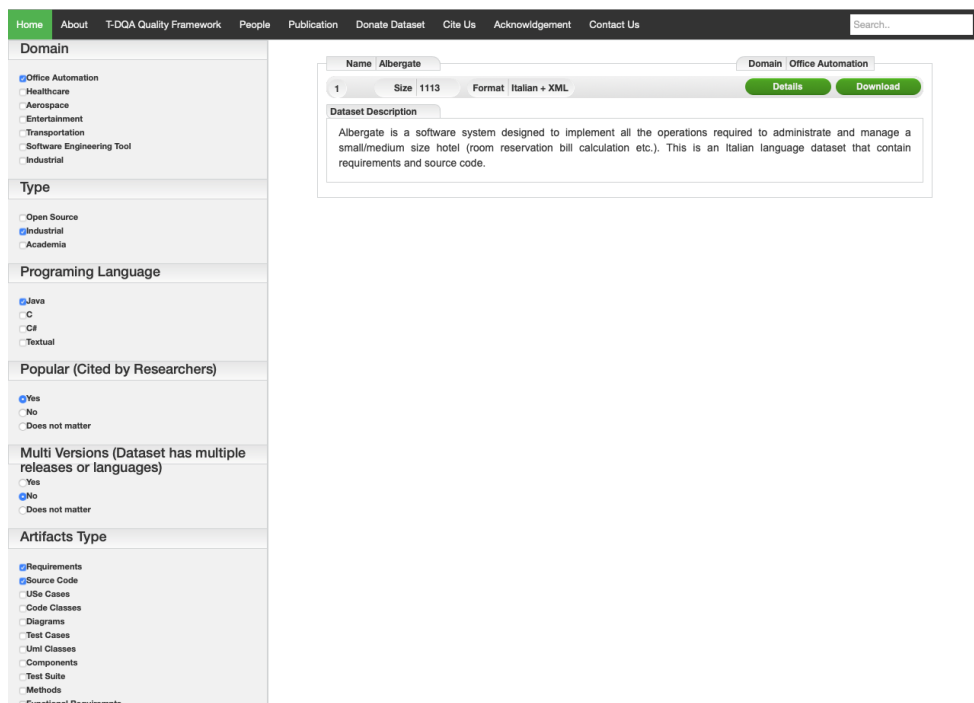


Figure 4.20: Dataset Search Example

## Chapter 5

# Conclusions

In this dissertation, we have presented and discussed the findings emerged from two preliminary studies that aimed to better comprehend the datasets used by traceability researchers and the feasibility of automatically generating such datasets by utilizing large scale open-source software. In the first study, we have conducted a large-scale systematic literature review to assess the current state of software traceability datasets that have been used by researchers in the community over the past fifteen years. In addition, we introduce a Traceability-Dataset Quality Assessment (T-DQA) framework aimed to provide researchers with a mean to categorize software traceability datasets and assist them to select an appropriate dataset for their research. Additionally, we conducted a study to better understand how requirements traceability researchers reason about datasets quality and choose among various available datasets and to assess our quality framework (T-DQA). To achieve this goal, we conducted an on-line survey that solicited feedback from 23 software traceability experts. The responses were analyzed systematically using grounded theory approach. As a result of this study, eleven types of dataset quality attributes were identified. For each quality attribute we identified a set of metrics used by researchers to quantify the quality attribute. Furthermore, we compared our finding with an existing traceability dataset quality framework. This framework was adapted from other application domain. Our analysis showed that experts agree with the dimensions of this framework, however, there are other attributes that expert use to reason about datasets quality which can augment this framework. As a result, we proposed *T-DQA v.2* which complements T-DQA based on the feedback collected from the traceability experts. *T-DQA v.2* can be used to characterize existing benchmark traceability datasets and enable the researchers to better reason about the quality of these datasets within the context of their research problems at hand.

Secondly, to investigate the feasibility of automatically generating traceability datasets we have conducted an empirical study and novel techniques that advances previous work. The proposed work introduced new approaches based on (i) *Web-Mining* and (ii) *Big-Data Analysis* to automate the creation of traceability datasets.

# Bibliography

- [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](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](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] Steve Adolph, Wendy Hall, and Philippe Kruchten. Using grounded theory to study the experience of software development. *Empirical Software Engineering*, 16(4):487–513, Aug 2011.
- [7] N. Ali, Y. Gueheneuc, and G. Antoniol. Trust-based requirements traceability. In *Proc. of ICPC*, pages 111–120, June 2011.
- [8] N. Ali, Y. Gueheneuc, and G. Antoniol. Trustrace: Mining software repositories to improve the accuracy of requirement traceability links. *TSE*, 39(5):725–741, May 2013.
- [9] N. Ali, Z. Sharafi, Y. Gueheneuc, and G. Antoniol. An empirical study on requirements traceability using eye-tracking. In *Proc. of ICSM*, pages 191–200, 2012.
- [10] Preethu Rose Anish, Balaji Balasubramaniam, Jane Cleland-Huang, Roel Wieringa, Maya Daneva, and Smita 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.
- [11] G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia. Information retrieval models for recovering traceability links between code and documentation. In *Proc. of ICSM*, pages 40–49, 2000.

- [12] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *TSE*, 28(10):970–983, Oct 2002.
- [13] V. Arnaoudova, L. M. Eshkevari, M. D. Penta, R. Oliveto, G. Antoniol, and Y. G. Guéhéneuc. Repent: Analyzing the nature of identifier renamings. *IEEE Transactions on Software Engineering*, 40(5):502–532, May 2014.
- [14] Venera Arnaoudova, Massimiliano Di Penta, and Giuliano Antoniol. Linguistic antipatterns: What they are and how developers perceive them. *Empirical Software Engineering (EMSE)*, 21(1):104–158, 2015.
- [15] Felix Bachmann, Len Bass, and Mark Klein. *Deriving Architectural Tactics: A Step Toward Methodical Architectural Design*. Technical Report, Software Engineering Institute, 2003.
- [16] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison Wesley, 2003.
- [17] G. Bavota, A. De Lucia, R. Oliveto, A. Panichella, F. Ricci, and G. Tortora. The role of artefact corpus in lsi-based traceability recovery. In *TEFSE, 2013*, pages 83–89, 2013.
- [18] G. Bavota, A. De Lucia, R. Oliveto, and G. Tortora. Enhancing software artefact traceability recovery processes with link count information. *Information and Software Technology*, 56(2):163–182, 2014.
- [19] George W. Beeler, Jr. and Dana Gardner. A requirements primer. *Queue*, 4(7):22–26, September 2006.
- [20] Pooyan Behnamghader, Duc Minh Le, Joshua Garcia, Daniel Link, Arman Shahbazian, and Nenad Medvidovic. A large-scale study of architectural evolution in open-source software systems. *Journal of Empirical Software Engineering (EMSE)*, 22(3):1146–1193, 2017.
- [21] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.
- [22] M. Borg, O.C.Z. Gotel, and K. Wnuk. Enabling traceability reuse for impact analyses: A feasibility study in a safety context. In *TEFSE*, pages 72–78, 2013.
- [23] Markus Borg, Per Runeson, and Anders Ardö. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *EMSE*, 19(6):1565–1616, 2014.
- [24] Michael Franklin Bosu and Stephen G MacDonell. A taxonomy of data quality challenges in empirical software engineering. In *Proc. of the Australian Software Engineering Conference (ASWEC)*, pages 97–106. IEEE, 2013.
- [25] Elke Bouillon, Patrick Mäder, and Ilka Philippow. A survey on usage scenarios for requirements traceability in practice. In Joerg Doerr and Andreas L. Opdahl, editors, *Requirements Engineering: Foundation for Software Quality*, pages 158–173, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

- [26] F. Bouquet, E. Jaffuel, B. Legeard, F. Peureux, and M. Utting. Requirements traceability in automated test generation: Application to smart card software validation. *SIGSOFT Softw. Eng. Notes*, 30(4):1–7, May 2005.
- [27] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
- [28] Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, 80(4):571 – 583, 2007.
- [29] Carla E. Brodley. Addressing the selective superiority problem: Automatic algorithm/model class selection, 1993.
- [30] Horst Bunke et al. Generation of synthetic training data for an hmm-based handwriting recognition system. In *null*, page 618. IEEE, 2003.
- [31] Matthieu Caneill, Daniel M. Germán, and Stefano Zacchiroli. The debsources dataset: two decades of free and open source software. *Journal of Empirical Software Engineering*, 22(3):1405–1437, 2017.
- [32] 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, December 2003.
- [33] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella. On the role of the nouns in ir-based traceability recovery. In *Proc. of ICPC*, pages 148–157, 2009.
- [34] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella. Traceability recovery using numerical analysis. In *Proc. of WCRE*, pages 195–204, Oct 2009.
- [35] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella. Improving ir-based traceability recovery via noun-based indexing of software artifacts. *SEP*, 25(7):743–762, 2013.
- [36] M. Cataldo, A. Mockus, J.A. Roberts, and J.D. Herbsleb. Software dependencies, work dependencies, and their impact on failures. *IEEE Transactions on Software Engineering (TSE)*, 35(6):864–878, November 2009.
- [37] Xiaofan Chen and John Grundy. Improving automated documentation to code traceability by combining retrieval techniques. In *Proc. of ASE*, pages 223–232, 2011.
- [38] J. Cleland-Huang, R. Settini, Chuan Duan, and Xuchang Zou. Utilizing supporting evidence to improve dynamic requirements traceability. In *Proc. of RE*, pages 135–144, 2005.
- [39] J. Cleland-Huang, R. Settini, Xuchang Zou, and P. Solc. The detection and classification of non-functional requirements with application to early aspects. In *Proc. of RE*, pages 39–48, 2006.
- [40] Jane Cleland-Huang, Adam Czauderna, Marek Gibiec, and John Emenecker. A machine learning approach for tracing regulatory codes to product specific requirements. In *Proc. of ICSE - Volume 1*, pages 155–164, 2010.



- [41] Jane Cleland-Huang, Adam Czauderna, Marek Gibiec, and John Emenecker. A machine learning approach for tracing regulatory codes to product specific requirements. In *ICSE (1)*, pages 155–164, 2010.
- [42] Jane Cleland-Huang, Orlena Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. Software traceability: trends and future directions. In *Proc. of the on Future of Software Engineering (FOSE)*, pages 55–69, 2014.
- [43] Jane Cleland-Huang, Orlena Gotel, Jane Huffman Hayes, Patrick Mader, and Andrea Zisman. Software traceability: Trends and future directions. In *Proc. of the 36th International Conference on Software Engineering (ICSE), India*, 2014.
- [44] Jane Cleland-Huang, Orlena Gotel, and Andrea Zisman. *Software and Systems Traceability*. Springer Publishing Company, Incorporated, 2012.
- [45] Jane Cleland-Huang, Orlena CZ Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. Software traceability: trends and future directions. In *Proc. of the on Future of Software Engineering*, pages 55–69, 2014.
- [46] Jane Cleland-Huang, Raffaella Settini, Oussama BenKhadra, Eugenia Berezhanskaya, and Selvia Christina. Goal-centric traceability for managing non-functional requirements. In *Proceedings of the 27th International Conference on Software Engineering, ICSE '05*, pages 362–371, New York, NY, USA, 2005. ACM.
- [47] Jane Cleland-Huang, Raffaella Settini, Xuchang Zou, and Peter Solc. Automated detection and classification of non-functional requirements. *Requir. Eng.*, 12(2):103–120, 2007.
- [48] Jacob Cohen. *Statistical power analysis for the behavioral sciences*. 1988.
- [49] Christopher S. Corley, Nicholas A. Kraft, Letha H. Etzkorn, and Stacy K. Lukins. Recovering traceability links between source code and fixed bugs via patch analysis. In *Proc. of the TEFSE*, pages 31–37, 2011.
- [50] Barthélémy Dagenais and Martin P. Robillard. Recovering traceability links between an api and its learning resources. In *Proc. of ICSE*, pages 47–57, 2012.
- [51] T. Dasgupta, M. Grechanik, E. Moritz, B. Dit, and D. Poshyvanyk. Enhancing software traceability by automatically expanding corpora with relevant documentation. In *Proc. of ICSM*, pages 320–329, 2013.
- [52] DataTypes.net. The most recent filename extension database, 2018.
- [53] Alan Davis, Scott Overmyer, Kathleen Jordan, Joseph Caruso, Fatma Dandashi, Anhtuan Dinh, Gary Kincaid, Glen Ledebor, Patricia Reynolds, Pradip Sitaram, et al. Identifying and measuring quality in a software requirements specification. In *Software Metrics Symposium, 1993. Proceedings., First International*, pages 141–152. IEEE, 1993.
- [54] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella. Improving ir-based traceability recovery using smoothing filters. In *Proc. of ICPC*, pages 21–30, 2011.

- [55] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. Enhancing an artefact management system with traceability recovery features. In *Proc. of ICSM*, pages 306–315, 2004.
- [56] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. Adams re-trace: A traceability recovery tool. In *Proc. of CSMR*, pages 32–41, 2005.
- [57] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. Can information retrieval techniques effectively support traceability link recovery? In *Proc. of ICPC*, pages 307–316, 2006.
- [58] A. De Lucia, R. Oliveto, and P. Sgueglia. Incremental approach and user feedbacks: a silver bullet for traceability recovery. In *Proc. of ICSM*, pages 299–309, 2006.
- [59] A. De Lucia, R. Oliveto, and G. Tortora. IR-based traceability recovery processes: An empirical comparison of ”one-shot” and incremental processes. In *Proc. of ASE*, pages 39–48, 2008.
- [60] A. De Lucia, R. Oliveto, and G. Tortora. The role of the coverage analysis during ir-based traceability recovery: A controlled experiment. In *Proc. of ICSM*, pages 371–380, 2009.
- [61] Andrea De Lucia, Rocco Oliveto, and Genoveffa Tortora. Assessing ir-based traceability recovery tools through controlled experiments. *EMSE*, 14(1):57–92, 2009.
- [62] M. Di Penta, S. Gradara, and G. Antoniol. Traceability recovery in rad software systems. In *Proc. of the International Workshop on Program Comprehension (IWPC)*, pages 207–216, 2002.
- [63] D. Diaz, G. Bavota, A. Marcus, R. Oliveto, S. Takahashi, and A. De Lucia. Using code ownership to improve ir-based traceability link recovery. In *Proc. of ICPC*, pages 123–132, 2013.
- [64] Hyunsook Do, Sebastian Elbaum, and Gregg Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Journal of Empirical Software Engineering (EMSE)*, 10(4):405–435, 2005.
- [65] Susan T Dumais. Latent semantic analysis. *Annual review of information science and technology*, 38(1):188–230, 2004.
- [66] Robert Dyer, Hridesh Rajan, Hoan Anh Nguyen, and Tien 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.
- [67] A. Egyed, F. Graf, and P. Grunbacher. Effort and quality of recovering requirements-to-code traces: Two exploratory experiments. In *Proc. of RE*, pages 221–230, 2010.
- [68] A. Egyed and P. Grunbacher. Automating requirements traceability: Beyond the record replay paradigm. In *Proc. of ASE*, pages 163–171, 2002.

- [69] G. Gates. The reduced nearest neighbor rule (corresp.). *Information Theory, IEEE Transactions on*, 18(3):431–433, May 1972.
- [70] V. Gervasi and D. Zowghi. Supporting traceability through affinity mining. In *Proc. of RE*, pages 143–152, 2014.
- [71] M. Gethers, R. Oliveto, D. Poshyvanyk, and A.D. Lucia. On integrating orthogonal information retrieval methods to improve traceability recovery. In *Proc. of ICSM*, pages 133–142, 2011.
- [72] A. Ghabi and A. Egyed. Code patterns for automatically validating requirements-to-code traces. In *Proc. of ASE*, pages 200–209, 2012.
- [73] GHTorrent. Downloads 2014-01-02, 2018.
- [74] Marek Gibiec, Adam Czauderna, and Jane 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.
- [75] B. G. Glaser and A. L. Strauss. The discovery of grounded theory: Strategies for qualitative research. *Transaction Books*, 2009.
- [76] Barney G Glaser. *Basics of grounded theory analysis: Emergence vs forcing*. Sociology press, 1992.
- [77] Michael W. Godfrey and Qiang Tu. Evolution in open source software: A case study. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 131–142, 2000.
- [78] Georgios Gousios. The ghtorrent dataset and tool suite. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*, pages 233–236, Piscataway, NJ, USA, 2013.
- [79] Georgios Gousios and Andy Zaidman. A dataset for pull-based development research. In *Proceedings of the Conference on Mining Software Repositories (MSR)*, MSR 2014, pages 368–371, 2014.
- [80] Hongyu Guo and Herna L Viktor. Learning from imbalanced data sets with boosting and data generation: the databoost-im approach. *ACM SIGKDD Explorations Newsletter*, 6(1):30–39, 2004.
- [81] Jin Guo, J. Cleland-Huang, and B. Berenbach. Foundations for an expert system in domain-specific traceability. In *Proc. of RE*, pages 42–51, 2013.
- [82] Jin Guo, Natawut Monaikul, Cody Plepel, and Jane Cleland-Huang. Towards an intelligent domain-specific traceability solution. In *Proc. of ASE*, pages 755–766, 2014.
- [83] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

- [84] Kevin A Hallgren. Computing inter-rater reliability for observational data: an overview and tutorial. *Tutorials in quantitative methods for psychology*, 8(1):23, 2012.
- [85] J. H. Hayes, G. Antoniol, B. Adams, and Y. G. Guéhéneuc. Inherent characteristics of traceability artifacts less is more. In *Proc. of RE*, pages 196–201, 2015.
- [86] Jane Huffman Hayes, Alex Dekhtyar, Jody Larsen, and Yann-Gaël Guéhéneuc. Effective use of analysts’ effort in automated tracing. *Requirements Engineering*, 23(1):119–143, Mar 2018.
- [87] J.H. Hayes, A. Dekhtyar, and J. Osborne. Improving requirements tracing via information retrieval. In *Proc. of RE*, pages 138–147, 2003.
- [88] J.H. Hayes, A. Dekhtyar, S.K. Sundaram, and S. Howard. Helping analysts trace requirements: an objective look. In *Proc. of RE*, pages 249–259, 2004.
- [89] E.A. Holbrook, J.H. Hayes, and A. Dekhtyar. Toward automating requirements satisfaction assessment. In *Proc. of RE*, pages 149–158, 2009.
- [90] GitHub Inc. Github, 2018.
- [91] K. Jaber, B. Sharif, and Chang Liu. A study on the effect of traceability links in software maintenance. *Access, IEEE*, 1:726–741, 2013.
- [92] Matthias Jarke and Yannis Vassiliou. Data warehouse quality: A review of the dwq project. In *IQ*, pages 299–313, 1997.
- [93] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European conference on machine learning*, pages 137–142, 1998.
- [94] Michael G Kahn, Marsha A Raebel, Jason M Glanz, Karen Riedlinger, and John F Steiner. A pragmatic framework for single-site and multisite data quality assessment in electronic health record-based clinical research. *Medical care*, 50, 2012.
- [95] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. An in-depth study of the promises and perils of mining github. *Journal of Empirical Software Engineering*, 21(5):2035–2071, 2016.
- [96] Shahedul Huq Khandkar. Open coding. *University of Calgary*, 23:2009, 2009.
- [97] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. Technical report, EBSE Technical Report EBSE-2007-01, 2007.
- [98] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. Morgan Kaufmann, 1995.
- [99] W. K. Kong and J. H. Hayes. Proximity-based traceability: An empirical validation using ranked retrieval and set-based measures. In *Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 45–52, 2011.

- [100] W.-K. Kong, J. Huffman Hayes, A. Dekhtyar, and J. Holden. How do we trace requirements: An initial study of analyst behavior in trace validation tasks. In *Proc. of the International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 32–39, 2011.
- [101] Rita Kovac, Yang W Lee, and Leo Pipino. Total data quality management: The case of iri. In *IQ*, pages 63–79, 1997.
- [102] Hongyu Kuang, P. Mader, Hao Hu, A. Ghabi, Liguang Huang, Lv Jian, and A. Egyed. Do data dependencies in source code complement call dependencies for understanding requirements traceability? In *Proc. of ICSM*, pages 181–190, 2012.
- [103] Richard J Landis and Gary G Koch. The measurement of observer agreement for categorical data. *Biometrics*, pages 159–174, 1977.
- [104] Wenbin Li, J.H. Hayes, Fan Yang, K. Imai, J. Yannelli, C. Carnes, and M. Doyle. Trace matrix analyzer (tma). In *Proc. of TEFSE*, pages 44–50, 2013.
- [105] Yonghua Li and J. Cleland-Huang. Ontology-based trace retrieval. In *Proc. of TEFSE*, pages 30–36, 2013.
- [106] Gernot A. Liebchen and Martin Shepperd. Data sets and data quality in software engineering. In *Proc. of the International Workshop on Predictor Models in Software Engineering (PROMISE)*, pages 39–44, 2008.
- [107] Gernot A. Liebchen and Martin 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.
- [108] Sugandha Lohar, Sorawit Amornborvornwong, Andrea Zisman, and Jane Cleland-Huang. Improving trace accuracy through data-driven configuration and composition of tracing features. In *Proc. of ESEC/FSE*, pages 378–388. ACM, 2013.
- [109] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella. Applying a smoothing filter to improve ir-based traceability recovery processes: An empirical investigation. *Information and Software Technology*, 55(4):741–754, 2013.
- [110] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *TOSEM*, 16(4), September 2007.
- [111] P. Mader, O. Gotel, and I. Philippow. Rule-based maintenance of post-requirements traceability relations. In *Proc. of RE*, pages 23–32, 2008.
- [112] A. Mahmoud. An information theoretic approach for extracting and tracing non-functional requirements. In *Proc. of RE*, pages 36–45, 2015.
- [113] A. Mahmoud and N. Niu. Source code indexing for automated tracing. In *Proc. of the TEFSE*, pages 3–9, 2011.

- [114] A. Mahmoud and N. Niu. Supporting requirements to code traceability through refactoring. *REJ*, 19(3):309–329, 2013.
- [115] S. Malviya, M. Vierhauser, J. Cleland-Huang, and S. Ghaisas. What questions do requirements engineers ask? In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 100–109, Sept 2017.
- [116] Vijay V Mandke and Madhavan K Nayar. Information integrity—a structure for its definition.
- [117] Vijay V. Mandke and Madhavan K. Nayar. Information integrity: A structure for its definition. In *Conf. on Information Quality*, pages 314–338, 1997.
- [118] Larry M Manevitz and Malik Yousef. One-class svms for document classification. *Journal of machine Learning research*, 2(Dec):139–154, 2001.
- [119] A. Marcus and J. I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proc. of ICSE*, pages 125–135, 2003.
- [120] Alan Matsumura and Nadia Shouraboura. Competing with quality information. In *IQ*, pages 72–86, 1996.
- [121] Brian W Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)*, 2(405):442–451, 1975.
- [122] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48, 1998.
- [123] Michael McCandless, Erik Hatcher, and Otis Gospodnetic. *Lucene in Action, Second Edition: Covers Apache Lucene 3.0*. Manning Publications Co., Greenwich, CT, USA, 2010.
- [124] Collin McMillan, Denys Poshyvanyk, and Meghan Revelle. Combining textual and structural analysis of software artifacts for traceability link recovery. In *Proc. of TEFSE*, pages 41–48, 2009.
- [125] Jane Cleland-Huang Mehdi Mirakhorli. Detecting, tracing, and monitoring architectural tactics in code. *IEEE Trans. Software Eng.*, 2015.
- [126] M. Mirakhorli. Preserving the quality of architectural decisions in source code, PhD Dissertation, DePaul University Library, 2014.
- [127] M. Mirakhorli and J. Cleland-Huang. Detecting, tracing, and monitoring architectural tactics in code. *TSE*, 42(3):205–220, March 2016.
- [128] Mehdi Mirakhorli and Jane 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.

- [129] Mehdi Mirakhorli and Jane Cleland-Huang. Detecting, tracing, and monitoring architectural tactics in code. *Transactions on Software Engineering (TSE)*, 42(3):205–220, 2016.
- [130] Mehdi Mirakhorli, Patrick Mäder, and Jane 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.
- [131] Mehdi Mirakhorli, Yonghee Shin, Jane Cleland-Huang, and Murat Cinar. A tactic centric approach for automating traceability of quality concerns. In *International Conference on Software Engineering, ICSE (1)*, 2012.
- [132] Luis Carlos Molina, Lluís Belanche, and Àngela Nebot. Feature selection algorithms: A survey and experimental evaluation. In *Proceedings of the 2002 IEEE International Conference on Data Mining, ICDM '02*, pages 306–, Washington, DC, USA, 2002. IEEE Computer Society.
- [133] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. Curating github for engineered software projects. *Journal of Empirical Software Engineering (EMSE)*, 22(6):3219–3253, 2017.
- [134] L.G.P. Murta, A. Van Der Hoek, and C.M.L. Werner. Archtrace: Policy-based support for managing evolving architecture-to-implementation traceability links. In *Proc. of ASE*, pages 135–144, 2006.
- [135] Sunil Nair, Jose Luis De La Vara, and Sagar Sen. A review of traceability research at the requirements engineering conference re@ 21. In *Proc. of RE*, pages 222–229, 2013.
- [136] N. Niu and A. Mahmoud. Enhancing candidate link generation for requirements tracing: The cluster hypothesis revisited. In *Proc. of RE*, pages 81–90, 2012.
- [137] Nan Niu, T. Bhowmik, Hui Liu, and Zhendong Niu. Traceability-enabled refactoring for managing just-in-time requirements. In *Proc. of RE*, pages 133–142, 2014.
- [138] A. N. Oppenheim. *Questionnaire Design, Interviewing and Attitude Measurement*. Pinter, 1992.
- [139] John O'Donoghue, Tom O'Kane, Joe Gallagher, Garry Courtney, Abdur Aftab, Aveline Casey, Javier Torres, and Philip Angove. Modified early warning scorecard: the role of data/information quality within the decision making process. *Electronic Journal Information Systems Evaluation Volume*, 14(1), 2011.
- [140] S. Pandanaboyana, S. Sridharan, J. Yannelli, and J.H. Hayes. Requirements tracing on target (retro) enhanced with an automated thesaurus builder: An empirical study. In *Proc. of TEFSE*, pages 61–67, 2013.
- [141] A. Panichella, A. De Lucia, and A. Zaidman. Adaptive user feedback for ir-based traceability recovery. In *Proc. of SST*, pages 15–21, 2015.

- [142] A. Panichella, C. McMillan, E. Moritz, D. Palmieri, R. Oliveto, D. Poshyvanyk, and A. De Lucia. When and how using structural information to improve ir-based traceability recovery. In *Proc. of CSMR*, pages 199–208, 2013.
- [143] Maria Luiza C. Passini, Katusca B. Estébanez, Graziela P. Figueredo, and Nelson 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.
- [144] D. Port, A. Nikora, J. H. Hayes, and L. Huang. Text mining support for software requirements: Traceability assurance. In *Proc. of the Hawaii International Conference on System Sciences*, pages 1–11, 2011.
- [145] T. Punter, M. Ciolkowski, B. Freimut, and I. John. Conducting on-line surveys in software engineering. In *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.*, pages 80–88, Sept 2003.
- [146] A. Qusef, G. Bavota, R. Oliveto, A. De Lucia, and D. Binkley. Scotch: Test-to-code traceability using slicing and conceptual coupling. In *Proc. of ICSM*, pages 63–72, Sept 2011.
- [147] A. Qusef, G. Bavota, R. Oliveto, Andrea De Lucia, and D. Binkley. Evaluating test-to-code traceability recovery methods through controlled experiments. *SEP*, 25(11):1167–1191, 2013.
- [148] A. Qusef, G. Bavota, R. Oliveto, A. De Lucia, and D. Binkley. Recovering test-to-code traceability using slicing and textual analysis. *JSS*, 88:147–168, 2014.
- [149] A. Qusef, R. Oliveto, and A. De Lucia. Recovering traceability links between unit tests and classes under test: An improved method. In *Proc. of ICSM*, pages 1–10, 2010.
- [150] M. P. Couper J. M. Lepkowski E. Singer R. M. Groves, F. J. Fowler Jr. and R. Tourangeau. *Survey Methodology*. 2nd ed. Hoboken, NJ, USA: Wiley, 2009.
- [151] M. Rahimi, M. Mirakhorli, and J. Cleland-Huang. Automated extraction and visualization of quality concerns from requirements specifications. In *Proc. of RE*, pages 253–262, 2014.
- [152] KR Remya and JS Ramya. Using weighted majority voting classifier combination for relation classification in biomedical texts. In *Proceedings of the International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, pages 1205–1209, 2014.
- [153] Gregorio Robles, Jesus M Gonzalez-Barahona, and Juan Julian Merelo. Beyond source code: the importance of other artifacts in software development (a case study). *Journal of Systems and Software*, 79(9):1233–1248, 2006.
- [154] Gregorio Robles, Jesus M Gonzalez-Barahona, and Juan Luis Prieto. Assessing and evaluating documentation in libre software projects. In *Workshop on Evaluation Frameworks for Open Source Software (EFOSS)*, 2006.



- [155] Colin Robson. *Real world research: a resource for users of social research methods in applied settings*. Wiley, Chichester, West Sussex, 3rd edition, 2011.
- [156] Gerard Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.
- [157] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.
- [158] Iván Santiago, Alvaro Jiménez, Juan Manuel Vara, Valeria De Castro, Verónica A Bollati, and Esperanza Marcos. Model-driven engineering as a new landscape for traceability management: A systematic literature review. *Information and Software Technology*, 54(12):1340–1356, 2012.
- [159] M. Shepperd, Q. Song, Z. Sun, and C. Mair. Data quality: Some comments on the nasa software defect datasets. *IEEE Transactions on Software Engineering*, 39(9):1208–1215, Sept 2013.
- [160] Emad Shihab, Zhen Ming Jiang, Walid M. Ibrahim, Bram Adams, and Ahmed E. Hassan. Understanding the impact of code and process metrics on post-release defects: A case study on the Eclipse project. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 4:1–4:10, 2010.
- [161] Yonghee Shin, J.H. Hayes, and J. Cleland-Huang. Guidelines for benchmarking automated software traceability techniques. In *Proc. of SST*, pages 61–67, 2015.
- [162] Janice Singer, Susan E. Sim, and Timothy C. Lethbridge. *Software Engineering Data Collection for Field Studies*, pages 9–34. Springer London, London, 2008.
- [163] David 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.
- [164] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.
- [165] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. Grounded theory in software engineering research: a critical review and guidelines. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*, pages 120–131. IEEE, 2016.
- [166] Anselm L Strauss. *Qualitative analysis for social scientists*. Cambridge University Press, 1987.
- [167] H. Sultanov and J.H. Hayes. Application of reinforcement learning to requirements engineering: requirements tracing. In *Requirements Engineering Conference (RE), 2013 21st IEEE International*, pages 52–61, July 2013.
- [168] Senthil Karthikeyan Sundaram, Jane Huffman Hayes, and Alexander Dekhtyar. Baselines in requirements tracing. In *Proc. of the Workshop on Predictor Models in Software Engineering (PROMISE)*, pages 1–6, 2005.

- [169] Kai Tian, Meghan Revelle, and Denys Poshyvanyk. Using latent dirichlet allocation for automatic categorization of software. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*, pages 163–166, 2009.
- [170] SL Ting, WH Ip, and Albert HC Tsang. Is naive bayes a good classifier for document classification. *International Journal of Software Engineering and Its Applications*, 5(3):37–46, 2011.
- [171] Irvine University of California. The sourcerer project. [sourcerer.ics.uci.edu](http://sourcerer.ics.uci.edu).
- [172] Bülent Üstün, W. J. Melssen, and Lutgarde M C Buydens. Facilitating the application of support vector regression by using a universal pearson vii function based kernel. *Chemometrics and Intelligent Laboratory Systems*, 81(1):29–40, 2006.
- [173] Christopher Vendome, Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Daniel German, and Denys Poshyvanyk. Machine learning-based detection of open source license exceptions. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 118–129, 2017.
- [174] Richard Y Wang and Diane M Strong. Beyond accuracy: What data quality means to data consumers. *Journal of management information systems*, 12(4):5–33, 1996.
- [175] Elaine J Weyuker, Thomas J Ostrand, and Robert M Bell. Comparing the effectiveness of several modeling methods for fault prediction. *Journal of Empirical Software Engineering (EMSE)*, 15(3):277–295, 2010.
- [176] D. Randall Wilson and Tony R. Martinez. Reduction techniques for instance-based learning algorithms. *Mach. Learn.*, 38(3):257–286, March 2000.
- [177] Suresh Yadla, Huffman Jane Hayes, and Alex Dekhtyar. Tracing requirements to defect reports: an application of information retrieval techniques. *ISSE*, 1(2):116–124, 2005.
- [178] Aiko Yamashita and Leon Moonen. Do developers care about code smells? - an exploratory survey. In *Proceedings of the Working Conference on Reverse Engineering (WCRE)*, pages 242–251, 2013.
- [179] Hsin yi Jiang, T.N. Nguyen, Ing-Xiang Chen, H. Jaygarl, and C.K. Chang. Incremental latent semantic indexing for automatic traceability link evolution management. In *Proc. of ASE*, pages 59–68, 2008.
- [180] R. K. Yin. Case study research: Design and methods. In *Sage*, 2009.
- [181] Yijun Yu, Jan Jurjens, and J. Mylopoulos. Traceability for the maintenance of secure software. In *Proc. of ICSM*, pages 297–306, 2008.
- [182] Xuchang Z., R. Settini, and J. Cleland-Huang. Phrasing in dynamic requirements trace retrieval. In *Proc. of COMPSAC*, volume 1, pages 265–272, 2006.
- [183] Roozbeh Zarei, Alireza Monemi, and Muhammad Nadzir Marsono. Automated dataset generation for training peer-to-peer machine learning classifiers. *Journal of Network and Systems Management*, 23(1):89–110, 2015.

- [184] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Sören Auer. Quality assessment for linked data: A survey. *Semantic Web*, 7(1):63–93, 2016.
- [185] He Zhang, Muhammad Ali Babar, and Paolo Tell. Identifying relevant studies in software engineering. *Information and Software Technology*, 53(6):625–637, 2011.
- [186] Jiaxin Zhu, Minghui Zhou, and Audris 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.
- [187] Waleed Zogaan, Ibrahim Mujhid, Joanna C. S. Santos, Danielle Gonzalez, and Mehdi Mirakhorli. Automated training-set creation for software architecture traceability problem. *Empirical Software Engineering*, pages 1–35, 2016.
- [188] Waleed Zogaan, Palak Sharma, Mehdi Mirakhorli, and Venera Arnaoudova. Datasets from fifteen years of automated requirements traceability research: Current state, characteristics, and quality. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 110–121, 2017.
- [189] X. Zou, R. Settini, and J. Cleland-Huang. Improving automated requirements trace retrieval: a study of term-based enhancement methods. *EMSE*, 15(2):119–146, 2009.

# Appendices

## Appendix A

# Traceability Datasets Quality Survey

## Traceability Datasets Quality Survey

### Introduction

Please read the following information about this research survey:

The goal of this study is to understand how traceability researchers assess the quality of the datasets they are using. Also, we want to know if the context is important for your assessment of quality and how context is impacting your decision. During the survey, you will be asked to answer a mix of both short answer and multiple-choice questions.

We anticipate that the survey would take less than 15 minutes of your time. We would very much appreciate it if you agree to participate in the survey. The deadline for the survey is January 31st (one week). Please let us know if you need more time.

This survey study is going to be conducted online and will not require any personal information. Also, the responses are going to be used for analysis only. Your participation in this study voluntarily. This means that you can decide to leave the study session at any time without penalties. If you have questions, concerns, or complaints about this study or you want to get additional information or provide input about this research, please contact Waleed Zogaan ([waz7355@rit.edu](mailto:waz7355@rit.edu)) or Dr. Mehdi Mirakhorli ([mxmvse@rit.edu](mailto:mxmvse@rit.edu)). If you have questions about your rights as a research subject you may contact Heather, RIT's Associate Director of Human Subjects Research Office (HSRO) at (585)475-7673 or by email at [hmfsrc@rit.edu](mailto:hmfsrc@rit.edu). You may also contact RIT's Office of Research Protections if:

- Your questions, concerns, or complaints are not being answered by the research team.
- You cannot reach the research team.
- You want to talk to someone besides the research team. You may keep or print this information for your records.

**Click YES to participate in the study**

- Yes
- No

**Q1. What is the country you work in?**

**Q2. Where do you work?**

- Academia
- Industry
- Other:

**Q3. What is your occupation?**

- Researcher
- Software engineer
- Other:

**Q4. Number of years of experience with software traceability?**

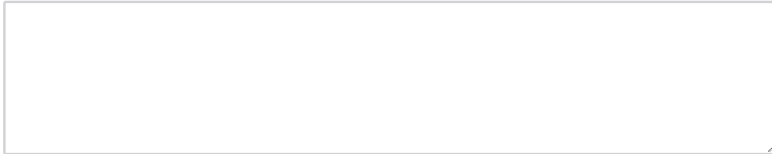
**Q5. What are the quality attributes you are looking for when you select datasets for your project? Please explain.**

**Q6. Does the application domain of your research project has an impact on what type of quality attributes you are looking for in a dataset? Please explain.**

**Q7. What dataset qualities have an impact on the meaningful conclusions being drawn from a research project? Please explain.**

**Q8. What are the datasets quality-attributes that could impact the generalizability of your research results?**

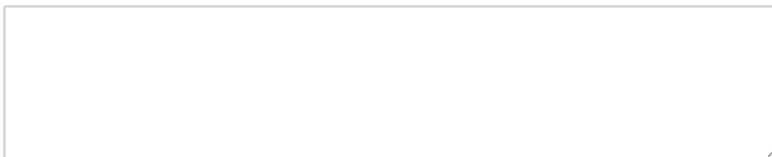




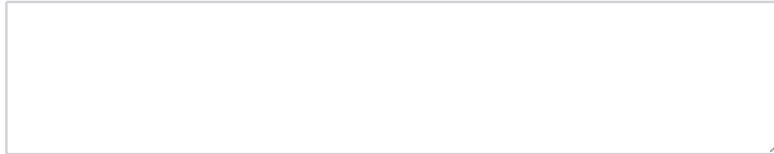
**Q9. Does the granularity of the tracing artifacts (Requirements, Source code, Use-cases, etc.) impact the type or direction of your research? By granularity we mean the existence of artifacts at different levels of detail, e.g., source files mapped to requirement documents versus source code methods mapped to individual requirements.**



**Q10. When do you consider that a dataset is trustworthy?**



**Q11. Do you consider academic or OSS projects sufficient enough for development or validation of a proposed research technique?**



**Q12. How important to you are the following as a measurement of your dataset quality?**

	Not important	Slightly Important	Moderately Important	Important	Very Important	Don't have an opinion
Dataset size	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Dataset domain	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Artifacts types	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The availability of answer-set.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The correctness of the answer-set.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The completeness (all required information is present) of datasets artifacts.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The representation (Dataset format) of the dataset.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

The dataset type (Academia, OSS, or Industry).	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The dataset collection method (Manually, automatically generated, or provided by collaborator )	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Trustworthiness (information is accepted to be correct, true, real, and credible)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Representation (Dataset format)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

**Comments:.**

**Q12. How important to you are the following attributes when you are searching for a dataset.**

	Not Important	Slightly Important	Moderately Important	Important	Very Important	Don't have an opinion
Dataset quality	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Dataset size	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Dataset domain	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Source of the data (industry, academia, or OSS)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Artifacts types	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Availability (the extent to which data is present, obtainable and ready for use )	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Trustworthiness (information is accepted to be correct, true, real, and credible)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Representation (Dataset format)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

**Comments:**

**Block 1**

. This is the end of the survey. Please click on "Submit" to submit all your answers or click on "Back" to make any changes to your answers.