Monthes: A Compact Software Traceability Dataset

Adhatus S. Ahmadiyah, Siti Rochimah and Daniel Siahaan
Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia

Keywords: Class Diagram, Code, Monthes Dataset, Requirements, Traceability

Abstract: Software artifact tracing from various software development phases has been utilized to maintain software quality. It leads to the popularity of the development of traceability tools to substitute manual labor tracing, which is prone to error. Along with the growth of software traceability tool development, traceability dataset needs are rising. Exclusively a few realistic traceability datasets are available for public access, guiding a practical degree of traceability tools for real projects. The scarcity is primarily due to the time and effort required to create the dataset. This paper presents efforts in developing a new traceability dataset from the education domain, namely Monthes. It involved artifact extraction and tracing activities from an established thesis monitoring application that professionals developed. Apart from assembling three artifacts, i.e., requirements, class diagram, and source codes, it results in three sets of traceability ground truths: requirement-to-code, requirement-to-design, and design-to-code. The software artifacts and ground truths would help researchers test the performance of their traceability tools or enhanced methods involving three phases of software development.

1 INTRODUCTION

Software traceability research covers the development and maintenance of traceability on software development artifacts. Since software artifacts are significant in measuring software quality, software quality can be determined and monitored by tracing all development phases. Initially, the tracing process was performed manually. Later, traceability tools replace it to tackle manual tracing drawbacks.

Some efforts have been made to develop traceability tools such as RETRO.NE (Hayes et al., 2018), ADAMS Re-Trace (de Lucia et al., 2005), TraceLab (Keenan et al., 2012), Poirot (Lin et al., 2006), TraCter (Mahmoud & Niu, 2011), and The SEOSS 33 (Rath, 2019). Despite its benefits, there are challenges in requirement traceability research. The challenges can be categorized into the following: time and effort costs, trouble preserving traceability during change, varying perspectives on traceability held by diverse project stakeholders, organizational issues and politics, and limited tool support (Saiedian, 2009).

For the advancement of automated software traceability research, datasets are essential. Tool evaluation requires realistic datasets to ensure these tools can enhance industrial project tracing results. The shortage of datasets for this type of research becomes a crucial restraining element. Such datasets are expensive to acquire and labor-intensive to gather and validate manually. In general, one of the most common challenges for academics in the software engineering field has been acquiring such software development datasets (Zogaan et al., 2017).

There are ranging of reasons why datasets are limited. As observed by (Saiedian, 2009), the first is that the effort of creating one is laborious, specifically in determining the ground truth that specifies the actual trace link between source and target artifacts. Second, authentic project artifacts are rarely made available. Third, businesses frequently need to find a way to share software development artifacts with the public since artifacts are considered proprietary. Fourth, academic researchers receive little acknowledgement for their work in establishing datasets.

Even though several traceability datasets are available (Zogaan et al., 2017), those were developed a few years ago. Moreover, newer ones are limited, and realistic traceability datasets from industrial projects are still desired.

This paper introduces a traceability dataset, called Monthes, from an actual project in the education domain. The dataset contains a list of requirement

Ahmadiyah, A., Rochimah, S. and Siahaan, D.
Monthes: A Compact Software Traceability Dataset.
DOI: 10.5220/0012105700003680
In *Proceedings of the 4th International Conference on Advanced Engineering and Technology (ICATECH 2023)*, pages 55-61
ISBN: 978-989-758-663-7; ISSN: 2975-948X

statements, design classes, source codes, and a collection of trace links. Tracing was performed from requirements statement to codes, requirements statement to design classes, and design classes to source codes. This research is a part of more extensive research in developing a new traceability approach using Property Listing Task (Ahmadiyah et al., 2022).

The rest of the paper is constructed as follows: the overview of the dataset is provided in Section II, its creation procedure is explained in Section III, potential research areas it may be used are presented in Section IV, and threats to the validity of the dataset are discussed in Section V. Finally, Section VI concludes research findings.

2 DATASET OVERVIEW

Monthes is an industrial project developed by a professional web development team. Specifically, it is a web-based thesis monitoring system for university students in Indonesia. The system covers proposal submission, thesis meetings, evaluation, grading, and reporting. The system involves multi-users: students, lecturers, examiners, and staff (administrator). Monthes was developed using the MVC architecture style using the Laravel framework.

Monthes dataset comprises requirement statements, source codes, and a class diagram extracted directly from the Monthes application. These artifacts were collected because each is the most frequently traced software artifact at their respective development phase: analysis, design, and implementation (Charalampidou et al., 2021). The dataset is also equipped with trace link ground truths between requirements and design; requirements and code; and design to code. The dataset is accessible to the public at Figshare website via https://doi.org/10.6084/m9.figshare.21582714.

2.1 Requirement Artifact

The requirement artifact contains a list of 17 functional requirements created by the Monthes development team in 2022. Monthes is used by four actors: administrator, student, advisor, and examiner. The functional requirements and their associated key terms are tabulated in Table 1. Key terms represent part of a sentence: subject, action, and object. A key term or combination of key terms is essential to help navigate to corresponding design and implementation artifacts in the tracing process.

Table 1: Monthes Functional Requirements.

Code	Requirement	Key Terms
FR01	Administrator adds user	administrator, add, user
FR02	Administrator manages schedule	administrator, manage, schedule
FR03	Administrator manages proposal upload	administrator, manage, proposal, upload
FR04	Administrator manages proposal meeting	administrator, manage, proposal, meeting
FR05	Administrator manages RMK data	administrator, manage, rmk, data
FR06	Administrator posts news	administrator, post, news
FR07	Student submits proposal	student, submit, proposal
FR08	Student revises proposal	student, revise, proposal
FR09	Student submits reports	student, submit, report
FR10	Advisor approves reports	advisor, approve, report
FR11	Lecturer adds question banks	lecturer, add, question, bank
FR12	Examiner approves reports	examiner, approve, report
FR13	Examiner grades reports	examiner,grade, report
FR14	Advisor grades reports	advisor, grade, report
FR15	Advisor adds question banks	advisor, add, question, bank
FR16	Advisor assigns the question to a student	Advisor, assign, question, student
FR17	Lecturer manages the lecture subject	Lecturer, manage, lecture, subject

2.2 Design Artifact

Monthes design artifact representing object abstraction, its structure, and its relationship to other objects was collected as a class diagram. The class diagram consists of 22 classes divided into three stereotypes, i.e., model, view, and controller classes following the Model-View-Controller architecture style. All class relationships are association types. In total, 63 methods and 125 attributes are distributed into three concerns. The statistics for each concern are tabulated in Table 2.

2.3 Implementation Artifact

Implementation artifact comes as source codes. Based on the class diagram, source codes were implemented using the Laravel framework. Each implementation class was grouped under controller, model, or view folder. There are 13 controllers under the app/Http/Controllers folder to accommodate application logic, nine models under the app/Models folder to handle data, and 32 views under the resources/views folder to accommodate the user interface. The recap of the implemented classes, methods, and attributes is displayed in Table 3.

Table 2: Class Diagram Statistics.

Concern	# Methods	# Attributes
Model	8	32
View	21	36
Controller	34	57

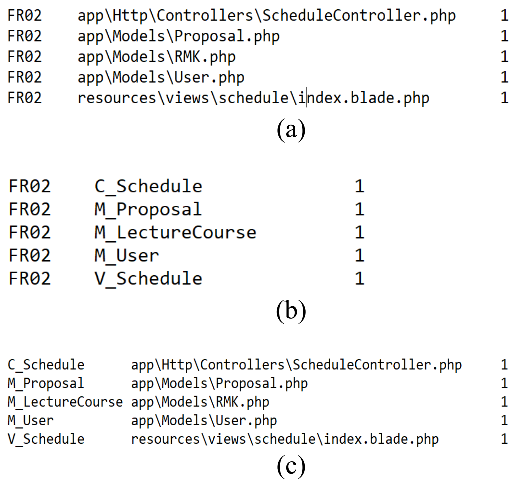
Table 3: Class Code Statistics.

Category	# Classes	# Methods	# Attributes
Controller	13	73	-
Model	9	19	18
View	32	-	-

2.4 Ground Truth

Trace link ground truth is provided in three files: requirement-to-design, requirement-to-code, and design-to-code. Each contains pairs of source and target artifacts. Trace link pairs are displayed as source artifact, target artifact, and link label.

Figure 1 shows trace link ground truth: (a) from the requirement to code, (b) from the requirement to design, and (c) from design class to code for requirement statement FR02: Administrator manages schedule. The reading for three ground truths is the same; specifically, the left column represents source artifact, the middle column represents target artifact, and the right column represents a link between both artifacts (1= link found, 0= no link). For example, from Figure 1a we know that requirement FR02 is implemented by five source code files. Meanwhile, we also aware that FR02 is related to five design classes (Figure 1b) and each design class is implemented to which source code files (Figure 1c).



```
FR02    app\Http\Controllers\ScheduleController.php     1
FR02    app\Models\Proposal.php                         1
FR02    app\Models\RMK.php                              1
FR02    app\Models\User.php                             1
FR02    resources\views\schedule\index.blade.php        1
```

(a)

```
FR02    C_Schedule          1
FR02    M_Proposal          1
FR02    M_LectureCourse     1
FR02    M_User              1
FR02    V_Schedule          1
```

(b)

```
C_Schedule      app\Http\Controllers\ScheduleController.php     1
M_Proposal      app\Models\Proposal.php                         1
M_LectureCourse app\Models\RMK.php                              1
M_User          app\Models\User.php                             1
V_Schedule      resources\views\schedule\index.blade.php        1
```

(c)

Figure 1: Monthes trace link ground truth for FR02. (a) Requirement to code, (b) requirement to design class, and (c) design class to code.

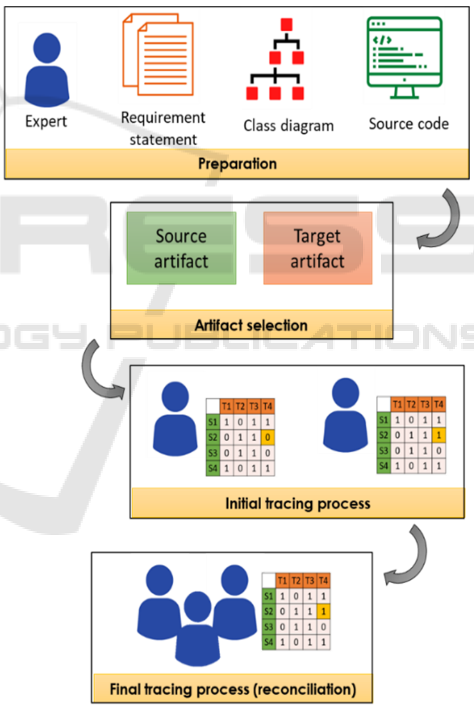


Figure 2: Monthes dataset creation process.

3 DATASET CONSTRUCTION

The overall process for generating the Monthes dataset is illustrated in Figure 2. It involved four main steps: preparation, artifact selection, initial tracing process, and final tracing process. Each process is discussed in the following subsections.

3.1 Preparation

First, we collected three artifacts (requirement statement, class diagram, and source code) from the Monthes development project. Requirement statement was taken from the original project by manually extracting functional requirements. Then, we translated it from Indonesian into English. The requirement statement was a simple statement containing an actor, action, and object. A class diagram was constructed using MVC architecture, in which each identified object was drawn in three stereotypes: boundary, control, and entity classes. The boundary class handles interaction between the system and users. The control class handles the business logic of the system. The entity class handles data. Code files were extracted from the development folder of the Monthes application. Since Monthes was developed using MVC, we consider source code files coming from three folders: the 'Controllers' folder is located under the app/Http folder, the 'Models' folder falls under the app folder, and the 'views' folder resides under the resources folder. The source code under the 'Controllers' folder implements business logic. Meanwhile, the 'Models' and 'views' folder contains source code implementing data and user interface.

Second, we invited experts consist of a system analyst, a programmer, and documentation personnel. The system analyst was previously involved in developing the Monthes application. Experts were involved in the tracing process (discussed in subsection 3.3 and 3.4).

Then, we checked experts and artifacts quality by employing seven traceability input factors from (Ali et al., 2012): expert domain knowledge, expert programming knowledge, ambiguous requirements, vague requirements, conflicting requirements, the granularity of source codes, and the identifier quality of source codes. As for the design artifact (class diagram), we checked different considerations about Model-View-Controller (MVC) architecture style. Figure 3 shows input (pink label), category of factors (yellow label), and factors for respective input types (blue label).

Ambiguity requirements resulted in different interpretations from experts. When this problem appears, a consensus is conducted to remove the ambiguity. Vague requirements are difficult to be interpreted correctly. This problem is also solved by agreement among project teams. Conflicting requirements resulted in incompatibility in or between artifacts. Conflicting requirements are avoided by finding possible requirement dependencies and then using them to locate potential conflicts.

Expert opinions contribute to the trustworthiness of the trace link, especially domain knowledge and programming language. Domain knowledge refers to specialist comprehension of the field in which the examined software system is being developed. In creating this dataset, the project team background with sufficient domain knowledge regarding student monitoring business processes was selected. In the meantime, programming knowledge characterizes expert capability to solve programming problems and code quality software in a specific programming language. The project team dealing with traceability has some experience in PHP language and Laravel framework.

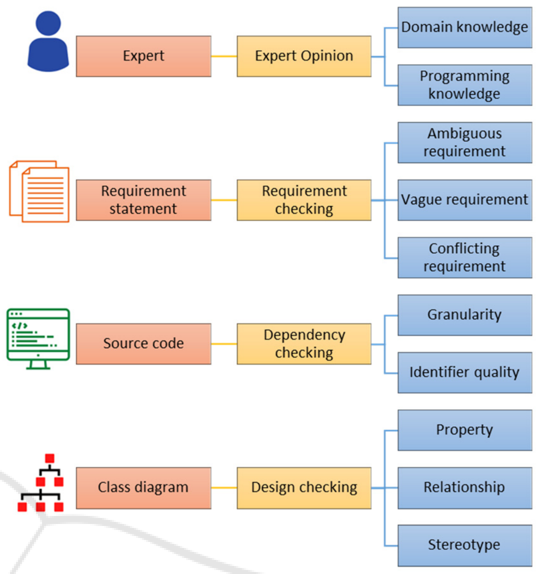


Figure 3: Traceability input factors.

Source code dictates the quality of traceability results through granularity level and identifier quality. In contrast, granularity demonstrates a detailed level of traceability (coarse, middle, or fine-grained). Coarse-grained traceability (tracing classes) is relatively less effort. Nevertheless, fine-grained traceability (tracing method) is more precise. The traceability team agrees on developing coarse-grained traceability to reduce the number of links from the requirement to code and design to code. In addition to the granularity level, identifier quality refers to the token's name in the source code, such as naming class, method, and attributes. Specific jargon usage in identifier names may affect the traceability result. We identified jargon usage in the class name, i.e., 'rmk', which stands for 'rumpun mata kuliah', an Indonesian term representing a group of university department majors.

As for the class diagram, we checked whether the class diagram design follows the MVC architecture style. Each class has different stereotypes: boundary, control, and entity. The Boundary class is concerned about user interaction and can only relate to controller classes. Meanwhile, a controller class handles the system's business logic and may relate to any class stereotypes. Lastly, the entity class handles data. An entity class can have a relationship to the controller class. In addition, we adopted class diagram elements identification used by (Fauzan et al., 2021), i.e., property and relationship.

3.2 Artifact Selection

Trace links are represented as pairs between source and target artifacts. In this step, we paired two out of three collected artifacts, resulting in three pairs for performing traceability. We performed forward tracing, in which the source artifact was obtained from the earlier development phase and target artifact was obtained from the latter.

3.3 Initial Tracing Process

For each traceability, two experts worked individually to create the initial traceability matrix from source artifact (S) to target artifact (T). Each expert listed each element of S and T, then they manually checked the relation between each pair. When lexical or semantic similarity is found, the pair is a match.

3.3.1 Establishing Requirement-to-Code Ground Truth

Initially, we created a coarse-grained traceability matrix from the requirement statement and class code. Two assigned experts worked individually to produce the traceability matrix by reviewing each requirement statement. Then experts found its match with source code files under the 'views', the 'Controllers', and the 'Models' folders. When a match is found, they put label 1, otherwise 0.

3.3.2 Establishing Requirement-to-Design Ground Truth

An initial traceability matrix presented as pairs between the requirement statement and design class were developed. Two experts worked individually and searched through each requirement statement in the class diagram to locate a match. Annotation was added to the trace link pairs—one to indicate a link, while zero means no link found.

3.3.3 Establishing Design-to-Code Ground Truth

At first, a traceability matrix was created to find the match between the source code and the class diagram. The granular level is considered coarse-grained since tracing between design classes was matched at the class level. We adopted tracing between two artifacts from (Katayama et al., 2018) with adjustments to fit MVC architecture styles. The experts explored each class design and source code folder individually to discover matches. When a link is discovered: it is annotated as one, otherwise zero.

3.4 Final Tracing Process

A reconciliation determines the final traceability results, mainly when different traceability matrix results between two experts exist. One author and existing experts collaborated to review the already gathered traceability matrix and develop a consensus on the final trace link. Finally, the traceability matrix was transformed into trace link pairs.

At the initial tracing process, each expert traceability matrix is displayed in Table 4 and Table 5. Since there is a difference regarding tracing to 'app\Models\Proposal.php', a reconciliation between experts is needed. One reconciliation example came from tracing FR11: Lecturer adds question banks to source codes. The reconciliation result (Table 6) shows no relation between FR 11: The lecturer adds question banks to source codes and 'app\Models\Proposal.php'.

In the traceability matrix (Table 5, and Table 6), S denotes the requirement statement, T denotes implementation, S1 denotes FR 11: Lecturer add question bank, while T1, T2, T3, T4, and T5 represent ‘app\Http\Controllers\QuestionController.php’, ‘app\Models\Question.php’, ‘app\Models\Proposal.php’, ‘app\Models\RMK.php’, and ‘resources\views\question\index.blade.php’ codes, respectively.

Table 4: Initial tracing of FR11 by expert #1.

	T1	T2	T3	T4	T5
S1	1	1	1	1	1

Table 5: Initial Tracing of FR11 by Expert #2.

	T1	T2	T3	T4	T5
S1	1	1	0	1	1

Table 6: Final Tracing of FR11.

	T1	T2	T3	T4	T5
S1	1	1	0	1	1

4 POTENTIAL RESEARCH TOPICS

Monthes dataset is a small dataset that depicts instances in the software development lifecycle when requirement artifacts have not kept up with software updates. It can be applied to traceability research areas, such as trace link establishment and enhancement.

Trace link establishment is research in developing trace links from software development artifacts. It is mainly used to monitor software quality throughout the development phases. For trace link establishment, the Monthes dataset could be utilized to propose an automated trace link method and derive performance measurements such as precision and recall. In addition, one might need to recover a missing artifact from a specific development phase, Monthes could be used to that extend.

Trace link enhancement focuses on updating existing trace links when there are software updates. In trace link enhancement, the Monthes dataset can be involved in proposing an enhancement method to show that the enhanced method outperforms the existing ones.

Since the Monthes dataset contains requirement-to-design, design-to-code, and requirement-to-code ground truth, it can also recover missing artifacts. For example, when the design artifact is not documented and we only have requirement and source code, we can reconstruct the missing class diagram and confirm it using requirement-to-design and design-to-code answer sets.

5 THREATS TO VALIDITY

Several threats potentially impact the validity of our Monthes dataset. We discussed a list of threats to validity associated with traceability dataset creation as adopted from (Zogaan et al., 2017) and additional internal threats to validity.

5.1 Data Acquisition

Selection bias comes from the non-representable population or does not fit the problem domain. In our case, this threat is avoided since we selected the Monthes application from the industrial project to develop a realistic traceability dataset.

Dataset equivalency threat concerns instances where researchers employ existing datasets to compare specific attributes of their datasets to demonstrate the chosen datasets' suitability. We directly used the project artifact to maintain its originality and ensure the granularity used for traceability without comparing its characteristics to other datasets.

Information bias threat reflects the accuracy of auto-generated datasets, misclassification, and data annotation. We minimized this threat by involving a system analyst and Monthes development team programmer to perform traceability.

Negative set bias is related to the rich and unbiased selection of outliers in training data, mainly in the classification problem. In developing the Monthes dataset, a reconciliation was conducted after the individual tracing process to escape negative set bias threats.

5.2 Trustworthiness

Trustworthiness concerns threats about the ground truth creation, student development, and peer-reviewed process. Specifically, the Monthes dataset trustworthiness threat was minimized by involving original developers to create the ground truths. Next is that our dataset was obtained from a thesis monitoring application developed by professionals, not a case study developed by students. The last one is that the trustworthiness threat was mitigated by conducting peer-reviewed reconciliation sessions in creating the ground truths.

5.3 Internal Validity

Internal validity measures how well a study reduces systematic bias or mistakes so that a causal conclusion can be made. In this dataset, it can be the learning bias threat of creating trace link pairs. To mitigate this threat, we had individual project teams create their initial traceability matrix. Then, reconciliation was held to decide the final link annotation.

5.4 External Validity

External validity relates to the extent to which the result of the dataset is generalizable. Currently, the dataset produced is the first generation. Therefore, there are limitations regarding generalization. The dataset could not generalize to all traceability tools and different project domains. However, since professionals developed it, dataset compactness could still represent industrial software projects. The dataset accommodates three artifacts that may not represent datasets containing other artifacts.

Regarding programming language threat, the dataset could generalize other datasets developed using object-oriented PHP and MVC.

6 CONCLUSIONS

We introduced a new dataset for traceability and discussed the method through which expert software developers built it. Our dataset is intended to aid in exploring various requirement-to-code, requirement-to-design, and design-to-code traceability research issues by the software engineering community.

In the future, the next generation of the Monthes dataset will extend the traceability granularity of requirement-to-code and design-to-code to fine-grained. This extension would allow trace up to the method level.

ACKNOWLEDGEMENTS

This work was supported by Institut Teknologi Sepuluh Nopember, Indonesia through the Department Research scheme (Skema Penelitian Dana Departemen) under grant no. 1732/PKS/ITS/2022.

REFERENCES

Ahmadiyah, A. S., Rochimah, S., & Siahaan, D. (2022). *Semantic Software Traceability Using Property Listing Task: Pilot Study*. 387–392. https://doi.org/10.1109/ies55876.2022.9888365

Ali, N., Guéhéneuc, Y. G., & Antoniol, G. (2012). Factors impacting the inputs of traceability recovery approaches. In *Software and Systems Traceability* (Vol. 9781447122395, pp. 99–127). Springer-Verlag London Ltd. https://doi.org/10.1007/978-1-4471-2239-5_5

Charalampidou, S., Ampatzoglou, A., Karountzos, E., & Avgeriou, P. (2021). Empirical studies on software traceability: A mapping study. *Journal of Software: Evolution and Process*, *33*(2). https://doi.org/10.1002/smr.2294

de Lucia, A., Fasano, F., Oliveto, R., & Tortora, G. (2005). ADAMS re-trace: A traceability recovery tool. *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*, 32–41. https://doi.org/10.1109/CSMR.2005.7

Fauzan, R., Siahaan, D., Rochimah, S., & Triandini, E. (2021). Automated Class Diagram Assessment using Semantic and Structural Similarities. *International Journal of Intelligent Engineering and Systems*, *14*(2), 52–66. https://doi.org/10.22266/ijies2021.0430.06

Hayes, J. H., Payne, J., & Dekhtyar, A. (2018). *The REquirements TRacing On Target (RETRO).NET Dataset; The REquirements TRacing On Target (RETRO).NET Dataset*. https://doi.org/10.5281/zenodo.1223649

Katayama, T., Mori, K., Kita, Y., Yamaba, H., Aburada, K., & Okazaki, N. (2018). *RETUSS: Ensuring Traceability System between Class Diagram in UML and Java Source Code in Real Time*.

Keenan, E., Czauderna, A., Leach, G., Cleland-Huang, J., Shin, Y., Moritz, E., Gethers, M., Poshyvanyk, D., Maletic, J., Hayes, J. H., Dekhtyar, A., Manukian, D., Hossein, S., & Hearn, D. (2012). TraceLab: An experimental workbench for equipping researchers to innovate, synthesize, and comparatively evaluate traceability solutions. *Proceedings - International Conference on Software Engineering*, 1375–1378. https://doi.org/10.1109/ICSE.2012.6227244

Lin, J., Lin, C. C., Cleland-Huang, J., Settimi, R., Amaya, J., Bedford, G., Berenbach, B., Khadra, O. ben, Duan, C., & Zou, X. (2006). *Poirot: A Distributed Tool Supporting Enterprise-Wide Automated Traceability*.

Mahmoud, A., & Niu, N. (2011). TraCter: A tool for candidate traceability link clustering. *Proceedings of the 2011 IEEE 19th International Requirements Engineering Conference, RE 2011*, 335–336. https://doi.org/10.1109/RE.2011.6051663

Rath, M. (2019). *The SEOSS 33 dataset - Requirements, bug reports, code history, and trace links for entire projects*. https://doi.org/10.7910/DVN/PDDZ4Q

Saiedian, H. (2009). *Why Software Requirements Traceability Remains a Challenge*. www.stsc.hill.af.mil

Zogaan, W., Sharma, P., Mirahkorli, M., & Arnaoudova, V. (2017). Datasets from Fifteen Years of Automated Requirements Traceability Research: Current State, Characteristics, and Quality. *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference, RE 2017*, 110–121. https://doi.org/10.1109/RE.2017.80