EVALUATION-ORIENTED ANALYSIS OF THE TRACEABILITY METHODS IN SOFTWARE SYSTEMS

Vojislav Tomašević*,
[0009-0000-5948-0123]

Jelica Protić,
[0000-0003-0846-0290]

Maja Vukasović
[0000-0003-0647-1922]

School of Electrical Engineering,
Belgrade, Serbia

Abstract:

Nowadays, software projects have become extremely large-scale, and their development requires the application of numerous standard and innovative technologies. Hence, the possibility of embedding and maintaining information about the mutual dependencies of the components in the software system becomes one of the key and necessary activities during development. This traceability feature enables the monitoring of the artifact connections and helps in preserving the system integrity. First, this paper briefly introduces the concept itself and the three most relevant groups of traceability methods: IR-based, ML-based, and graph-based. Then, the paper focuses on evaluation data from the available literature that provide an insight into the comparative performance of the approaches. The basic metrics observed were: standard (Precision, Recall, and F-score), and specific ones. Different evaluation setups and configurations are also considered. Although some papers indicate that ML-based methods can outperform IR-based ones, no general conclusion can be drawn because of quite different datasets and evaluation environments.

Keywords:

Traceability, Information Retrieval, Machine Learning, Graphs, Performance Evaluation.

INTRODUCTION

In software engineering, *traceability* refers to the existence of links between artifacts in complex software systems. More specifically, it is the ability to interconnect pairs of artifacts into a network, maintain links between them over time, and use the resulting network to answer various questions related to the software product and its structure. It helps with better understanding and easier analysis of a software system from different aspects [1]. According to [2], artifacts can be classified into three categories: a) artifacts related to programming code (code segments, compiled programs, microservices, scripts, tests, etc.), b) artifacts related to project management (requirements, models, benchmark programs, roadmaps, quality and management plans, risk assessment, etc.), and c) artifacts related to documentation (diagrams, various instructions, internal documentation, etc.).

Correspondence:
Vojislav Tomašević

e-mail:
vojkan99@gmail.com

DOI: 10.15308/Sinteza-2026-151-158

Nowadays, because of their ever-increasing size, software projects require traceability as one of the essential and inevitable activities during development. This activity also allows:

- a better grasp of relations between the components in a complex system;
- easier detection of deficiencies in the system, which raises the level of quality of the software product;
- impact analysis [3] which quickly and reliably identifies parts of the system that need to be modified in order for the system to adapt to an emerging change;
- efficient risk management while reducing project costs; and
- improved software security and safe system functioning.

In recent decades, numerous traceability methods have been proposed. Depending on their goal, the methods apply various techniques from different areas of software engineering and artificial intelligence, such as information retrieval (IR), machine learning (ML), data mining, natural language processing, graph theory, knowledge representation, etc. It is difficult to directly compare them due to their completely different system configurations and datasets [4], [5]. In order to regulate this area, some authors suggest certain standards, although they are mostly limited to individual aspects. E.g., it is proposed in [6] to create a general knowledge base only for traceability requirements. In addition to the lack of a common concept for the application of traceability in different domains, the problem of different terminology has also been noted (e.g., the term traceability is used for the ability to track system elements, as well as for links between them [7]). An attempt to unify terminology was proposed in [4].

For resolving the traceability issue, two approaches can be used:

1. When creating each new artifact, chronological data for all artifacts that led to its creation is saved; and
2. Links between artifacts are created later as a result of the analysis of the entire system.

The first approach requires upgrading the tools used in software development with mechanisms for automatic generation of artifacts and trace links, while the second approach relies on applying appropriate techniques for analysis of available data, often in combination with manual work. The focus of our paper is on the second approach because it is much more common in the open literature and practice.

This paper aims to briefly survey the main traceability approaches and, more specifically, shed more light on the evaluation issues (types of evaluation, metrics, comparative analysis). After this introduction, in section 2, the principles of the three most prevalent groups of traceability methods are presented. Section 3 describes standard and specific metrics used for performance evaluation and provides an overview of their application for the considered groups of methods. Section 4 defines the four types of evaluation analysis of traceability methods and presents the results of relevant studies for each type. Finally, a brief conclusion is given.

2. TRACEABILITY METHODS

Depending on the primary technique, traceability methods can be classified into several groups: IR based,

ML-based, graph-based, rule-based, ontology-based, probabilistic, heuristic, and hybrid methods. The first three groups, as the most popular ones, are presented below.

2.1. IR-BASED METHODS

In the context of solving the traceability problem, IR techniques are used to identify links between software artifacts based on their textual similarities. Although artifacts can have different structures, they generally contain textual parts that describe the informal semantics of the artifact (for example, comments in the source code, intuitively named variables) [8]. If two artifacts have a textual similarity, this indicates a possible connection between them. The higher the similarity, the more likely it is a candidate for the link.

The classic way of applying IR techniques for traceability is shown in Figure 1. First, two collections of artifacts (source and target) are defined. The task is to find the associated pairs of source and target artifacts. In order for IR techniques to work well, the text is normalized (conversion to lowercase letters, tokenization, removal of noise, i.e., numbers, symbols, stop words). The artifacts are represented using some model (VSM, Bag-of-Words, TF-IDF, etc.). Then, similarity is calculated for each pair (most often cosine, Jaccard index, Dice coefficient). The obtained results are ranked so that for each source artifact a list containing its similarities to all target artifacts is generated. It serves as a list of candidate links for that artifact (with calculated probabilities). Finally, relevant links are selected from the list of candidates. These are usually the first k candidates

from the list, or all candidates whose probability exceeds a defined threshold. The links are forwarded to an expert for validation and their final confirmation or rejection.

The issues in IR-based methods stem from the fact that software artifacts can be linked in different ways, and text is only one of them. Therefore, various strategies are introduced that include additional elements in the similarity calculation process. For example, the authors in [9] discuss the links between documentation (use cases) and source code (Java classes) and suggest using code ownership information. For each code component, the author is distinguished, and the context of the work is identified. A query that represents a use case is compared to classes using a standard IR technique. Also, the similarity of the query with the code author's context is calculated. Similarity scores of the classes whose authors also have the highest similarity of the context with the query are increased.

The performance of the IR-based method is affected by the existence of polysemy (multiple interconnected meanings of the same word) and synonymy (words with very similar meanings) in textual parts of software artifacts. To reduce their effect, various techniques are used, such as contextual models (LSA), Word embeddings (Word2Vec, GloVe), domain-specific dictionaries and ontologies, etc.

2.2. ML-BASED METHODS

In the last decade, with the rapid development of artificial intelligence, numerous ML techniques have been extensively used for traceability purposes. In [10], [11], a detailed overview of their application for requirements traceability is given. Supervised learning (SL) methods are the most often used, with 64.2%, according to [10]. These methods use datasets to train AI models with the goal of identifying existing patterns and connections from data. Then, the model created in this way is used to predict the correct output for new input data from a real environment.

In the traceability context, ML is used to automatically find patterns for linking artifacts, and to suggest or rank traceability links according to those patterns. The training set contains artifacts, features and labels defined over pairs of artifacts. The label indicates whether there is a link between the artifacts in the pair or not. The model learns from the combinations of features that are present in the cases when the link exists. The trained model is applied to the new artifact in three steps: 1. The new artifact is paired with all other artifacts, 2. Features are calculated for all generated pairs, and 3. Result for each pair is classified as a binary label ("link exists" or "no link").

Depending on the applied ML algorithm, the result for each pair can be binary (link exists/no link) or a link score, i.e., the probability that a link between the artifacts in a pair exists. If the result is a score, then it is compared with the given threshold. If it is greater than the threshold, it is considered that the pair is connected by a link; otherwise, it is not. As can be seen, in both cases, the final classification is binary. From the above, it can be concluded that ML-based traceability methods abandon the calculation of similarities as in IR-based methods and transform the generation of links into a classification problem.

In this approach, the link arises as a result of the classification of a pair of artifacts rather than of a single artifact. Unlike the usual applications of ML methods with object classification (objects are labeled), in the traceability context, we deal with the relation classification problem (links between artifacts are labeled).

The hardest problem with ML-based traceability methods is providing a data set for training the model because it must contain the pre-existing trace links. Solving this problem has been greatly facilitated in recent years with the popularization and development of open-source software, as numerous source code repositories have become available. They encompass a huge number of artifacts and their automatic tracking is usually necessary. However, in open-source projects,

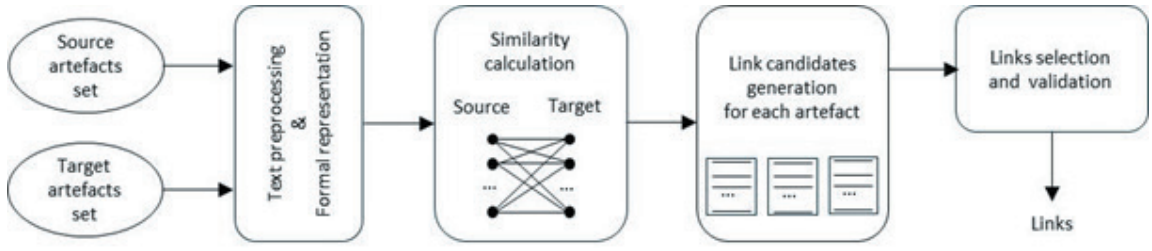

Figure 1. Traceability approach in IR-based methods

most artifacts refer to the source code, while some other types of artifacts are often missing. In [12], this problem was analyzed in detail for 383 open-source projects randomly selected from GitHub. It was noticed that the requirements and documentation-related artifacts were prevalently missing since they covered only 6.12% of the total number of artifacts. However, about 14.8% of the considered projects still contained these artifacts, which was sufficient for the analysis. Similar explorations were conducted in [11] for 26 studies that used 46 datasets, 65% of which were open-source.

According to some studies (e.g. [14]), ML-based traceability methods usually attain better results than IR-based ones for a sufficiently large training dataset since the calculation of text similarity is domain-independent, while the training dataset describes the domain. However, these claims must be taken vary cautiosly because of different benchmarks, evaluation methodologies, and setups. Sometimes, these two approaches are combined so that IR provides a better input for the ML-based approach.

2.3. GRAPH-BASED METHODS

An intuitive way to represent artifacts and the links between them is their visualization. It can be achieved by graphs whose nodes represent artifacts and whose edges represent traceability links. A graph-based approach usually exploits some information about the project that IR-based or ML-based methods do not use, such as information about the development history or about the system architecture. In this approach, links are generated not exclusively based on textual similarity, as is the case with IR-based methods, but also the structure of the entire graph is used. Graphical representation allows the discovery of indirect connections between artifacts.

Graph-based traceability procedure starts with generating a graph based on available data (existing links between artifacts, source code structure, UML relations, etc.), followed by an analysis of the graph using different algorithms to discover new links (with a certain probability). These algorithms examine the transitivity of artifacts (whether two artifacts are connected via other nodes in the graph), whether they belong to some cluster based on similarity, whether they have similar neighbors, etc. The most often used techniques for graph analysis are:

- propagation of links through the graph (Graph Traversal - searches for paths in the graph and establishes links between nodes on the same path, Random Walk - discovers hidden connections of artifacts with other nodes based on the probability of reaching those nodes during a random walk through the graph, etc.);
- link prediction (the algorithm recognizes global patterns in the graph structure and predicts new links between artifacts accordingly; for each pair of unconnected nodes, it estimates the prediction score using a set of metrics and, after comparing it with the threshold, declares the link as valid or invalid); and
- graph embeddings (algorithm learns the meaning of artifact representations; after mapping the nodes into the vector space, the vectors are compared by similarity, usually cosine, and if they are similar enough, a link is established between these nodes).

The problem with the graph-based approach arises when working with larger datasets. In these cases, the graphs, due to their size, become unmanageable and not scalable. In addition, the implementation of this approach is quite complex, and in order for the method to give satisfactory results, the data in the initial graph must be highly reliable.

3. EVALUATION METRICS

Given that all traceability methods have the same goal (the generation of links between artifacts created or used during software development), the performance of all methods is determined by the quality of the links and the speed of their generation. Various metrics are used for performance evaluation. Some of the metrics apply to all traceability methods, so they can be considered standard. The success rate of an individual method is evaluated by comparing its results with a manually validated set of real links, often represented by a traceability matrix. In all methods, it is common to determine the following parameters: *TP* (True Positives) - the number of real links that the method found, *FP* (False Positives) - the number of wrong links that the method found, and *FN* (False Negatives) - the number of real links that the method did not find. Then, the following standard metrics can be calculated:

$$Precision = TP/(TP + FP), \quad Recall = TP/(TP + FN),$$
$$F\text{-}score = 2 * (Precision * Recall)/(Precision + Recall)$$

These metrics are originally statistical classification performance metrics. The *Precision* measures how many of the found links actually exist (correct links), while *Recall* measures how many of all real links the method found. A high *Precision* means that few non-existing

links are found, and a high *Recall* means that few real links are not identified. *F-score* is the harmonic mean of *Precision* and *Recall* that represents a unique value as a balance between these two metrics. Besides these standard metrics, additional metrics can also be applied. An overview of the most common additional metrics by groups of traceability methods described in Section 2 is provided in Table 1.

IR-based methods usually generate a ranking list of link candidates, and the *k* best-ranked ones are considered as links. Therefore, the performance of these methods is often estimated by *Precision@k* and *Recall@k* metrics.

ML-based methods often employ supervised classification and are evaluated by classification metrics. Due to the extreme class imbalance, *PR-AUC* has been increasingly used, while *Accuracy*, one of the key classification metrics in the ML field, is avoided. The *Confusion Matrix* is not a real metric, but it provides a basis for applying other metrics.

4. EVALUATION OF METHODS

In order to provide some quantitative performance data, an extensive open literature search was carried out (more than forty papers) with a focus on recent evaluation studies from the last decade that report on post-hoc automated traceability recovery methods. The limiting factors for direct performance comparison were the lack of standardized datasets, and the wide diversity of evaluation frameworks, setups, and methodologies. Four types of evaluation of the traceability methods can be distinguished based on studies from the literature:

Type A. Evaluation of individual method for different datasets; in this way, the generality of the method is usually examined in terms of its application in different software projects;

Type B. Evaluation of one method in different configurations on the same dataset; the goal is to find a configuration that gives the best performance and to determine the impact of some configuration elements on the success of the method;

Type C. Evaluation of several different methods of the same group on the same dataset; it examines which method gives the best performance and for what reasons; and

Type D. Evaluation of several different methods of the different groups on the same dataset; it examines the impact of the different technologies on the traceability process; the standard metrics are used.

The examples of these types from open literature are given in the following (in many studies, they are combined).

Type C. The paper [13] compares two IR-based traceability recovery approaches, TAROT and AVIATE. TAROT connects artifacts not only based on their textual similarity, but also takes into account the structure of the software system. Namely, it starts from the fact that artifacts usually do not exist independently, but that instead many of them are in hierarchical or other relationships with other artifacts. Therefore, neighboring artifacts are also considered to influence link generation. AVIATE is designed for bilingual software projects (which are very common in the Internet era). The underlying idea is to enrich the existing textual contents

Table 1. Metrics Used in Traceability Methods

Methods	Standard metrics	Additional metrics	Description of additional metrics
IR-based	*Precision* *Recall* *F-score*	*PR-Curve*	Graphical representation of the trade-off between Precision and Recall for different thresholds
		Average Precision (AP)	Shows how the true links are arranged in the rankings for each artifact (whether they are highly ranked or not)
		Mean Average Precision (MAP)	Shows whether true link rankings are consistent across all artifacts (calculates average AP across all artifacts)
ML-based		*Area Under PR-Curve (PR-AUC)*	Shows how well the method ranks correct links over incorrect ones, regardless of the defined threshold
		Confusion Matrix	Gives good insight into the type of errors that are made when deciding whether a link exists or not
Graph-based		*Path Coverage*	Checks if all expected paths of related links exist from the first to the last artifact
		Graph Edit Distance	Determines the minimum number of changes that should be made to obtain a reference graph from the generated one
		Centrality-based evaluation	Performs structural evaluation of the graph (identifies key artifacts, bottlenecks, insufficiently connected parts)

of artifacts with multiple automatic translations of the words in them and thus increase the probability of generating the right link. A comparison of these approaches was carried out on a dataset with 17 bilingual open-source software projects from GitHub in English, Chinese, Korean, Japanese, and German. The VSM (Vector Space Modeling) IR model was used in this study.

The first experiment compared classic VSM (without translator) with VSM and one of four translators (NLLB 1.3B model, M2M-100-12B model, Google Translate, and Tencent Translate). It confirmed that links are established more successfully with a translator (*AP* for VSM with a translator improved over classic VSM in the range of 0.7 to 1.43, depending on the translator). Then, TAROT was included in VSM with a translator, which improved *AP* by 15.47% and *MAP* by 6.98%, considering all projects compared to VSM with a translator. An even better effect was obtained by including AVIATE in VSM with translator, 31.43% (*AP*) and 11.22% (*MAP*). These results show that in bilingual projects, both approaches, TAROT and AVIATE, bring improvement, but linguistic variability had a greater influence on traceability than the structure of the software system.

Type B and *Type D*. In [14], the authors propose an ML-based classifier for automatic generation and maintenance of traceability links called TRAIL (Traceability Link Classifier). This approach uses existing knowledge in the form of pre-existing traceability links to train a classifier, which then classifies a potential link between any two artifacts (new or existing) into two classes: valid or invalid. ML algorithms employ feature selection techniques to derive features from existing links, from which they recognize statistical patterns for association. Since the model is trained on a Cartesian product of an artifact set (each artifact is paired with each other), most of the links are invalid, and the classes are very imbalanced. Hence, the approach also requires the use of balancing techniques. The evaluation of the TRAIL method was performed on 11 datasets obtained from 6 software projects. There were 32616 possible links between artifacts, of which 7.97% were valid. Six classifiers were examined (k-Nearest Neighbors with k = 5, Näive Bayes, Logistic Regression, Random Forest, Support Vector Machine, and a Voting ensemble classifier which combines all other algorithms), 6 feature selection techniques (*cfs, correlation, gainRatio, infoGain, symmetrical*, and *none*), and 4 balancing techniques (*undersampling, smote*, 5050, and *none*). The evaluation had two goals:

1. Identifying the TRAIL baseline configuration (feature selection technique, balancing technique, and classifier) that gives the best performance in terms of *F-score* across all of the datasets (*Type B*),
2. Performance comparison of the TRAIL baseline configuration with 7 popular IR-based approaches (*Type D*).

In order to achieve the first goal, 144 possible combinations of the mentioned techniques were examined. The results showed that the highest average *F-score* (75.18%) was achieved with a combination of correlation-based feature selection, *smote* rebalancing, and a Random Forest classifier. Then, it was chosen as the baseline configuration for the evaluation. For all 24 combinations of the other two techniques, Random Forest gave better results than all other classifiers. For each classifier, Table 2 shows the combination that leads to its best *F-score*.

The results also show that the smote balancing technique significantly increases *Recall* (from 57% without balancing to 76% for the baseline configuration), while maintaining *Precision* of 75%. However, even though only slightly more than half of the valid links were found without balancing, *Precision* is still high at 86%. Therefore, when it is necessary to reduce the number of links to inspect, a high *Recall* approach is followed (ensuring a minimal number of missed valid links), while a high *Precision* approach is used (although some valid links will remain undetected) when more reliable link predictions are needed. For the baseline configuration, the impact of changing the feature selection technique was also analysed, and it revealed that there are no significant differences in the *F score* when the technique is changed and that the same performance can be achieved with a small number of features. The conclusion is that

Table 2. The Best Results of Six Classifiers

Classifier	Max F-score (%)	Feature selection technique	Rebalancing technique
Random Forest	75.18	*correlation*	*smote*
5-NN	63.10	*gainRatio/infoGain*	*smote*
Naive Bayes	40.17	*gainRatio/infoGain/symetrical*	*none*
Logistic Regression	61.00	*gainRatio/infoGain/symetrical*	*none*
Support Vector Machine	47.69	*symetrical*	*smote*
Voting ensemble	66.96	*infoGain*	*none*

the choice of classifier in TRAIL has the greatest impact (Random Forest was the best), and feature selection and balancing techniques can be used to favor *Precision* or *Recall* as needed.

In order to achieve the second goal, it was initially required to adjust the types of results of the TRAIL and IR-based approaches so that they could be compared. Since IR techniques generate ranked lists of link candidates, a classification was introduced that declares the first *k* candidates in the list as valid and the rest as invalid.

The evaluation was carried out by applying each of the IR-based methods to each of the 11 datasets, and then for each dataset, the method with the best *F-score* was selected. For 8 datasets, VSM was the best, and for the remaining 3 datasets, LSA, LM-Dirichlet, and BM25 outperformed. Then, these methods were compared to the TRAIL base configuration using standard metrics. TRAIL performed better on all datasets across all metrics. For 6 datasets, TRAIL had *Precision* and *Recall* above 70%, and for 3 datasets, even above 90%. The best result for *Precision* was 95.96%, and for *Recall*, 99.03%, and the lowest values for both metrics were around 56%. IR-based methods had *Precision* and *Recall* above 60% for one dataset, while the lowest values for both metrics were 30%. However, results varied across datasets. E.g., for two datasets, TRAIL improved less than 5%, while for 6 datasets it improved more than 30%. The biggest improvement (eAnci dataset) was over 40% for all three metrics. Therefore, the TRAIL base configuration outperformed the best IR-based methods for each dataset by 26% on average for all three metrics.

Type A. In [15], a semi-automatic graph-based method was proposed. Using XML patterns defined in the metamodel, it generates artifact links and trace type rules (describing the semantics of links) and stores them in a repository implemented in the open-source graph database Neo4j. This database supports the Cypher language of graph queries that allow analyzing the graph and finding potential new links. The developer decides which of the potential links will be added to the repository. The evaluation of the method was performed for 3 datasets. The study confirms that the graph database Neo4j provides faster query execution and better system performance than a relational database, but no evaluation data on traceability accuracy and quality of links can be found in this study.

Finally, although an extensive literature survey was carried out, the direct quantitative performance cross-evaluation with statistical meta-analysis was practically infeasible to conduct due to quite different dataset characteristics, evaluation environments, and evaluation metrics used in various studies. Still, some general qualitative comparison implies that the text-based methods (represented by IR-based) are widely used because of their simplicity, but are sensitive to textual patterns and linguistic issues (synonyms, polysemy). ML-based methods require proper training sets with correctly labelled data to achieve better quality of links. The advantage of the graph-based methods is the better modelling of complex relations due to structural context, but their scalability and complexity are major concerns. In order to inherit the best of three worlds, a promising approach is based on their synergy, where IR-based generates the candidates (list of links), graph-based improves their quality by filtering and expanding them, and ML-based finally ranks and confirms the candidates.

5. CONCLUSION

The analysis of the literature reveals that IR-based, followed by ML-based and graph-based traceability methods, are the most common in software engineering nowadays. Their goal is to build a linked structure of related artifacts that would allow a better understanding of the system and facilitate monitoring the impact of future changes. However, this structure is often very complex because, in addition to useful links, unnecessary or redundant links are also present, which makes the management of artifacts and links difficult and time-consuming. Therefore, an additional effort is necessary to enrich the types of links in order to precisely express the reasons for linking two artifacts. Predefining a set of types could enable interoperability of different types of traceability methods. Also, it was observed that most contemporary methods use standard performance metrics (*Precision, Recall*), but using more specific metrics could improve the evaluation quality. Some kind of standardized traceability evaluation methodologies, protocols, and datasets would be highly beneficial for the sake of direct comparison.

An important problem in contemporary security-critical software is to preserve its integrity from intruders. Despite their high potential, existing traceability techniques pay very little attention to this problem. Therefore, support of traceability methods in solving the problem of data security should certainly be one of the high-priority tasks, considering the serious consequences that may arise due to malicious changes of artifacts and their links.

REFERENCES

[1] G. Spanoudakis and A. Zisman, "Software traceability: a roadmap," in *Chang SK (ed) Advances in software engineering and knowledge engineering, recent advances*, World Scientific Publishing, 2005. doi:10.1142/9789812775245_0014

[2] "What is an artifact in software development?," [Online]. Available: https://www.techtarget.com/searchsoftwarequality/definition/artifact-software-development. [Accessed 16 February 2026].

[3] T. W. Aung, H. Huo and Y. Sui, "A Literature Review of Automatic Traceability Links Recovery for Software Change Impact Analysis," in *IEEE/ACM 28th International Conference on Program Comprehension (ICPC)*, 2020. doi:10.1145/3387904.3389251

[4] E. R. Batot, S. Gérard and J. Cabot, "A Survey-driven Feature Model for Software Traceability Approaches," in *25th International Conference on Fundamental Approaches to Software Engineering*, Munich, Germany, 2022. doi:10.1007/978-3-030-99429-7_2

[5] M. Seiler, P. Hübner and B. Paech, "Comparing traceability through information retrieval, commits, i nteraction logs, and tags," in *IEEE/ACM 10th International Symposium on Software and Systems Traceability (SST)*, 2019. doi:10.1109/SST.2019.00015

[6] A. M. Duarte, D. Duarte and M. Thiry, "TraceBok: Toward a software requirements traceability body of knowledge," in *IEEE 24th International Requirements Engineering Conference (RE)*, 2016. doi:10.1109/RE.2016.32

[7] G. Antoniol, J. Cleland-Huang and M. Vierhauser, "Grand Challenges of Traceability: The Next Ten Years," *ArXiv*, abs/1710.03129., 2017. doi:10.48550/arXiv.1710.03129

[8] A. De Lucia, A. Marcus, R. Oliveto and D. Poshyvanyk, "Information retrieval methods for automated traceability recovery," *Software and Systems Traceability*, pp. 71-98, 2012. doi:10.1007/978-1-4471-2239-5_4

[9] D. Diaz, G. Bavota, A. Marcus, R. Oliveto, S. Takahashi and A. D. Lucia, "Using code ownership to improve IR-based Traceability Link Recovery," in *21st International Conference on Program Comprehension (ICPC)*, 2013. doi:10.1109/ICPC.2013.6613840

[10] C. Xu, Y. Li, B. Wang and S. Don, "A systematic mapping study on machine learning methodologies for requirements management," *IET Software*, pp. 405-423, 2023. doi:10.1049/sfw2.12082

[11] X. Li, B. Wang, H. Wan, Y. Deng and Z. Wang, "Applications of Machine Learning in Requirements Traceability: A Systematic Mapping Study (S)," in *Shi-Kuo Chang, ed., 'SEKE', KSI Research Inc.*, 2023, p. 566–571. doi:10.18293/SEKE2023-135

[12] Y. Ma, S. M. Fakhoury, M. Christensen, V. Arnaoudova, W. A. Zogaan and M. Mirakhorli, "Automatic Classification of Software Artifacts in Open-Source Applications," in *15th International Conference on Mining Software Repositories*, Gothenburg, Sweden, 2018. doi:10.1145/3196398.3196446

[13] S. Kexin, R. Yiding, K. Hongyu, G. Hui, M. Xiaoxing, R. Guoping, S. Dong and Z. He, "AVIATE: Exploiting Translation Variants of Artifacts to Improve IR-based Traceability Recovery in Bilingual Software Projects," in *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24)*, New York, USA, 2024. doi:10.1145/3691620.3695023

[14] C. Mills, J. Escobar-Avila and S. Haiduc, "Automatic Traceability Maintenance via Machine Learning Classification," in *International Conference on Software Maintenance*, 2018. doi:10.1109/ICSME.2018.00045

[15] R. Elamin and R. Osman, "Implementing Traceability Repositories as Graph Databases for Software Quality Improvement," in *IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2018. doi:10.1109/QRS.2018.00040