Automatically recovering students' missing trace links between commits and user stories

Sivana Hamer, Christian Quesada-Lopez, and Marcelo Jenkins

Universidad de Costa Rica, San Pedro, Costa Rica
{sivana.Hamer,cristian.quesadalopez,macerlo.jenkins}@ucr.ac.cr

Abstract. Trace links between commits and user stories can be used in educational software engineering projects to track progress and determine the students' contribution to projects' requirements. Thus, traceability can be helpful in courses for grade assessment, and project monitoring and improvement. Currently developers, including students in courses, manually link commits and issues using version control systems (e.g., Git) and issue tracking systems (e.g., Jira). However, manual trace links are often incomplete. In our study, we found that only 43% of the commits are linked to stories in the analyzed project. Therefore, there is a need to automatically or semi-automatically create trace links. This study aims to automatically recover trace links between commits and user stories requirements in an undergraduate student project with twenty students and four teams. We used unstructured data from messages, code and files of commits and stories to gather textual similarity measures. We evaluated the effectiveness of information retrieval (Vector space model, Latent semantic indexing and BM25) and machine learning (Random forests, Decision trees and Neural networks) techniques in recovering missing links using textual and process data. Machine learning models outperformed information retrieval models in precision, recall, and F-measure. Machine learning models were able to effectively recover missing trace links with an average of 93% precision and 94% recall, showing the applicability of the approach.

Keywords: software engineering education · traceability · link recovery · information retrieval · machine learning · mining software repositories

1 Introduction

Traceability supports, throughout the development life cycle, different software engineering tasks and activities such as verification and validation, change impact analysis, change management, and requirement completeness analysis [1]. The establishment and use of trace links have been applied and studied in a wide range of software development environments [2–5] and software artifacts [6–8]. Traceability practitioners in software projects have reported commonly using tracing for tracking the implementation of requirements and tasks, finding the origin and rationale of requirements, documenting requirements' stories, and analyzing requirements coverage in source code [9]. These benefits could also be applied to software development projects in education.

Links between changes and issues (i.e., features, bugs or enhancements) can be used to build prediction or localization models, characterize the impact of a change, assign bugs to developers, and map source code and features [10]. A common practice to establish manual trace links between changes and issues is tagging [11]. Source code changes (i.e., commits) are saved in version control systems (VCS), including information about the who, what, why, when and how of a change [12]. Meanwhile, issue tracking systems (ITS) collect bugs, enhancement requests, features and tasks [13]. To link issues with changes, developers tag (in the change) the identification of the implemented issue [14].

Tracing commits with user story requirements in student's software engineering projects would allow instructors and students to monitor requirement progress and completion. Roadblocks with requirement implementation can then be detected to be tackled by students or instructors providing additional feedback. Traceability could help determine a student's or team's contribution and can be combined with other information such as process [15] and quality [16] metrics. Trends in feature implementation could also be determined to find recurring patterns in the process or product. Therefore, the information gathered could help instructors determine improvement opportunities for courses and help grade assessment based on data.

Despite the benefits of establishing manual trace links, it has been reported that links are frequently inaccurate and untrustworthy [17]. Several studies, both in open source software and industrial projects, have also indicated that links between commits and issues are often incomplete [10, 11, 18]. Therefore, there is a need to establish automatic or semi-automatic trace links.

In this paper, we aim to automatically recover trace links between commits and user stories in students' software engineering projects. We investigated missing link recovery in an undergraduate software engineering project with twenty students, 1,035 commits, and 230 committed or done stories. We analyzed to what extent students manually linked commits and user stories. Furthermore, we assessed a total of six information retrieval and machine learning techniques. Information retrieval methods used the messages, files and code of the commits and stories to create missing links. Additionally, machine learning techniques utilized both process and textual similarity information to recover the links.

The remainder of this paper is organized as follows. Section 2 describes related work. Section 3 details the methdology of our study. Section 4 presents and discusses the results. Section 5 describes the work's conclusions and future work.

2 Related Work

Several approaches have previously investigated recovering commits and issues.

Heuristic and information retrieval approaches have been previously applied. Śliwerski et al. [14] used syntactical and semantical heuristics to determine if a change fixed a bug. Wu et al. [19] applied ReLink, an algorithm that automatically learns from links between bugs and changes to recover missing links.

ReLink uses the time, ownership, and textual features to determine the links. Nguyen et al. [20] proposed MLink, an approach that calculates, with Tf-Idf, code and textual similarity between bugs and bug-fixing changes. Hübner and Paech [21] used data from interaction logs, requirements, and source code in student projects. Hübner and Paech [5] further extend their approach by using the issue identifiers provided in the commit messages. Our approach differs from most previous approaches as it focuses on another context, students' software engineering projects, and also includes probabilistic models.

Other approaches have used machine learning techniques. Sun et al. [22] proposed FRLink, an approach that included non-source code documents and discards irrelevant source code files. FRLink extracts the textual similarity for the commit code, non-source code and logs, and issue reports with the vector space model. Then, FRLink classifies if the commits and issues are related using a binomial prediction model. Sun et al. [23] further improved their approach by considering unlabeled data with PULink. Rath et al. [11] used process and text-related features with machine learning techniques to generate missing links between the commits and issues (bugs and improvements). They calculated the textual similarity between commits and issues using algebraic information retrieval models with the commits messages and files, and issues summaries and files. Ruan et al. [10] utilized deep learning techniques to capture the semantic representation between commits' logs and code, and issues titles, texts and code. We differ from previous machine learning approaches as we focus on validating the approach in another context and requirements.

Therefore, we extend previous studies by generating more evidence, comparing other techniques, and applying the approach in a less studied context.

3 Methodology

In this section, we describe the methodology of our study. Subsection 3.1 describes our approach. Subsection 3.2 details the analyzed project. Subsection 3.3 comments the threats to validity.

3.1 Approach

Figure 1 shows the approach used in this study with four main stages: data extraction, data preprocessing, information retrieval modeling, and machine learning modeling. Commits from the version control systems (VCS) and user stories from the issue tracking system (ITS) are first extracted from their repositories. Next, the true commit-story and false commit-story trace links are recovered. Then, the unstructured data from the commits and user stories are preprocessed. The preprocessed data is then used with information retrieval techniques to extract the textual similarity between artifacts and retrieve the most relevant documents. Finally, machine learning models are trained and evaluated. The complete approach has been implemented in Python. Each step will be further elaborated in the following paragraphs.

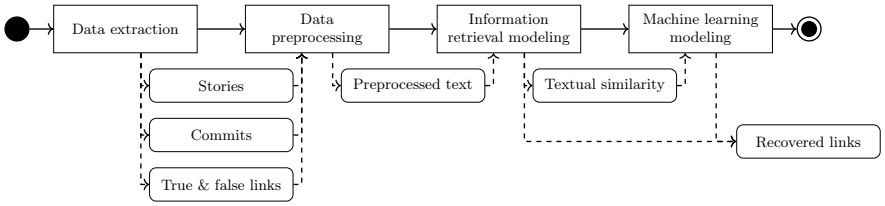

Fig. 1: Trace link recovery approach

Data extraction. Many different artifacts are created during a software development project. In our study, we used the commits and requirements written as user stories.

Commit. For Git [24] a commit is a snapshot, a version of the repository at that moment. Each commit has a unique identifier and saves the changes to the files with the associated code. Furthermore, other metadata of the commit is saved, such as when the changes were made, the developer who made the change, and why the change was made in the commit message (subject and body).

User story. A user story is a type of requirement used in agile software engineering written from the perspective of the end-user in natural language [25]. User stories were written based on the Connextra template (I as a *role* want *function* so that *business value*) [26]. Furthermore, the user story also included a summary, acceptance criteria, user story points, sprints, creation time, termination time, and creator. The stories were saved in Jira [1], but some additional information, such as the assignees of each story, was saved in Excel. We consider the message of a story the summary and the description. This included the Connextra template and acceptance criteria.

The relationship between our selected artifacts is shown in Fig. 2. The squares represent artifacts. Relationships with an arrow represent linking, while squares represent containment. Commits have messages, a pair of subjects and bodies. Stories also have messages, which include their title and description. A user story can have multiple associated linked commits, and vice versa. Furthermore, a commit contains files that were changed and can be modified in multiple commits. These files save numerous code changes of addition, deletion, or modification.

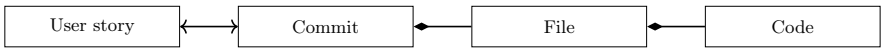

Fig. 2: Relationship between commits and stories

[1] https://www.atlassian.com/software/jira

For both commits and stories, we mapped each developer and team to a unique identifier. To map the commits, we used a mailmap [2]. Meanwhile, the stories are mapped with a program based on a configuration file that maps the students' emails to the unique identifier. For both the commits and stories, students are also mapped with their respective team with a configuration file.

Finally, the trace links between the commits and user stories were recovered using previously used heuristics [10]. Trace links in the project were saved by tagging in the commits' message the associated story identifier. Therefore, we recover the true links by identifying the story identifier in a commit message. Meanwhile, false links are created for each commit with a true link. For each story not associated with the linked commit, a false link is created. Furthermore, to reduce the potential false links only commits that were committed between 7 days before the creation or after the completion date of the story [23]. Furthermore, stories had to be either committed to a sprint or done to be included as either a true or false link.

Data preprocessing. To select more representative terms using information retrieval models, the commits' and stories' unstructured data is linguistically preprocessed [27]. Particularly, the stories messages, and the commits messages, files and source code changes between snapshots. To preprocess the text, text normalization, camel and snake case words splitting, text tokenization, stop words removal, and morphological root stemming were applied. Words were stemmed using the Porter stemmer algorithm [28]. The nltk [29], ftfy [3] and Cucco libraries [4] were used to preprocess the text. As the text could be written in either English or Spanish, all the text was translated to English with google translate [30]. To process the commit files, we also included the full path. Furthermore, the files' extensions were removed and folders were split. Finally, to extract relevant terms from the source code from the commit, we eliminate source code keywords for CS, SQL and JS files [31].

Information retrieval modeling. Information retrieval models select and rank documents based on queries, and are classified into four main categories: probabilistic, algebraic and logical, informational theoretic, and bayesian [32]. In our case, we used these models to gather and rank the textual similarity between commits and stories. Most traceability information retrieval-based approaches have focused on using algebraic models, specifically the Vector space model [33]. In this paper, both algebraic and probabilistic models were analyzed. The examined models were the Vector space model (VSM) [34], Latent semantic indexing (LSI) [35] and BM25 [36, 37]. We used Gensim's [38] implementation of algebraic models. For BM25, we used the library Rank-BM25 [5]. The default configuration is used for all the models.

[2] https://git-scm.com/docs/git-check-mailmap#_mapping_authors
[3] https://github.com/LuminosoInsight/python-ftfy
[4] https://github.com/davidmogar/cucco
[5] https://github.com/dorianbrown/rank_bm25

The story's message is treated as the document in the model, while the commit's message, files and code are the queries. We index each of the different documents and the query is then processed to obtain the commit-story similarity score. Based on the similarity score given by the models, the relevant documents for a commit are the two highest, non-zero stories. For the queries with multiple entries (i.e., code and files), a total score is calculated based on four different strategies: maximum score, median score, average score, and the average of the top 5 scores. Models performance was evaluated based on the precision, recall and F-measure [27]. Precision is the fraction of retrieved documents relevant, recall is the fraction of relevant documents retrieved, and F-measure is the harmonic mean for the precision and recall.

Machine learning modeling. Our machine learning approach has four major steps: data selection, training and evaluation. For the data selection phase, the commits, stories, true and false links, and commit-story similarity scores are gathered from the previous steps. Previously, process and textual similarity sets have been used [11]. Therefore, we divide our features into the following sets:

Process. Incorporates process related information. We capture: (p_1) committer; (p_2) committer team; (p_3) story assignees; (p_4) teams assigned to story; (p_5) committer is an assignee (committer == an assignee); (p_6) committer team is an assigned team (committer == an assigned team); (p_7) story created date <= commit date; and, (p_8) commit date <= story resolution date. Features p_5, p_6, p_7 and p_8 are binary.

Similarity. Contains the textual similarity scores. We used: (s_1) commit message and story message similarity; (s_2) commit files and story message similarity; (s_3) commit code and story message similarity. Since we have multiple similarity scores, we select the highest performing information retrieval models for each similarity feature.

All. All the features ($p_1, \cdots, p_8, s_1, s_2, s_3$) from both the process and similarity set are selected.

Previously decision trees (DT), neural networks (NN) and random forests (RF) models have been applied to recover commit-issue trace links [10, 11]. Therefore, we trained and investigated these three different supervised machine learning classifiers. We used Scikit-learn's [39] implementation of machine learning models. Decision trees and random forests apply default hyperparameters. Meanwhile, the neural network adopts all the default hyperparameters except iterations with a value of 500 to achieve model convergence. As the datasets were severely unbalanced, data was balanced with sub-sampling using Scikit-learn. In order to mitigate the effects of randomness while training the model, every model and set is tested ten times. The classifier returns if the commit-story pair was either linked or unlinked. We evaluated each model's precision, recall and F-measure as they have been previously used to assess the performance of machine learning models for commit-issue link recovery [10, 11, 22].

3.2 Software project

We analyzed a third-year undergraduate software engineering project from the Universidad de Costa Rica with Git and Jira repositories. Furthermore, students also detailed some aspects related to the user stories in an additional excel. The students applied agile methodologies and developed a medical attention mobile application using Blazor, C# and SQL Server. The project was developed by twenty students divided into four teams with scrums of scrums [6]. At the end of the project, students added 1,035 commits and 279 user stories in the backlog. 230 stories were committed to a sprint or done.

Students were required to manually tag while committing the identifier of the associated user stories and tasks. After the course ended, the number of manual links between the commits and user stories was calculated. Out of 279 user stories, 55 were linked to only one commit, 102 were linked to various commits, and 122 had no linked commits. Meanwhile, for the 1,035 commits, 353 were linked to only one story, 92 were linked to multiple stories, and 590 were not linked to stories. Therefore 56.3% of the stories were linked to commits, and 43% of the commits were linked to stories.

The number of commits with linked stories by student and team is shown in Fig. 3. Each bar represents the number of commits committed or stories assigned to the student or team. Each color of the bar represents the number of commits or stories that were not linked (0), linked to one artifact (1) or linked to multiple artifacts (n). There was significant variance in the number of commits linked to stories by students (Subfigure 3a) and teams (Subfigure 3b). The unlinked commits by students ranged from 22% to 95%. Furthermore, commits were not linked proportionally to their commits. For example, S14 had a total of 45 commits and had an average commit contribution. Despite not being a top contributor for commits, S14 had the lowest number of unlinked commits. In total, 16 students, representing 80% of the students, linked commits to more than one story. Thus it was a frequent practice to implement more than one functionality per commit. The differences of linking practices for commits are also present in the teams, with unlinked commits ranging from 37% to 83%. Meanwhile, the user stories assigned by students (Subfigure 3c) or teams (Subfigure 3d) had a higher degree of linking than commits, but still, there was considerable variability. For the assigned stories, 4% to 72% of stories were unlinked. The unlinked stories average was 33%. There were even two students who had linked at least one commit to their assigned stories. Still, there are 65 stories without an assignee. These unassigned stories were unlinked in 98% of the stories. 21% to 67% of the stories assigned to a team were not linked to a commit.

3.3 Threats to validity

There are several potential threats to the validity of our study. **External validity.** We studied an undergraduate software engineering project. A validity

[6] https://www.agilealliance.org/glossary/scrum-of-scrums

(a) Students' commits to stories

(b) Teams' commits to stories

(c) Students' stories to commits

(d) Teams' stories to commits

Fig. 3: Manual links of students and teams

threat is related to how generalizable our results are to other projects, contexts and technologies. Thus, we cannot claim the generalizability of our results. As future work, we plan to evaluate additional undergraduate software engineering projects. **Internal validity.** The manual links were generated by the students' during the project as experts of the project. Thus we cannot guarantee the accuracy of the links. In order to mitigate the influence of grading in creating the links, the evaluation of the trace links was conducted after the project was finished. Only committed or done stories were used to create automatic trace links. There is a validity threat with regards to the selection of the stories used in the linking. To mitigate this threat we included any story that was committed or done in either Jira or excel.

4 Results and discussion

In this section, we detail (Subsection 4.1) and discuss (Subsection 4.2) the results of our study.

4.1 Results

Figure 4 shows the precision, recall and F-measure for the information retrieval models and sets. The x-axis represents the different models. The y-axis represents the value for each performance measure. Each dot is a different result for the

pair of the sets and models. The code and file sets have multiple dots per model-set pair as there are different strategies to calculate the total score. The best performing model was BM25 with the commit message, achieving an average precision of 32% and recall of 50%. Across all information retrieval models, the message attained the best results with 30% precision and 46% recall on average. Meanwhile, the files and code had similar performance. On average, files had 7% precision and 11% recall, and code had 6% precision and 10% recall. Furthermore, there was no singular strategy for selecting the total for files and code that had outperformed the performance of the models. Across all the models and sets, the recall was higher than precision. However, though the information retrieval models had a low performance, they were useful as they provided textual data for the machine learning models.

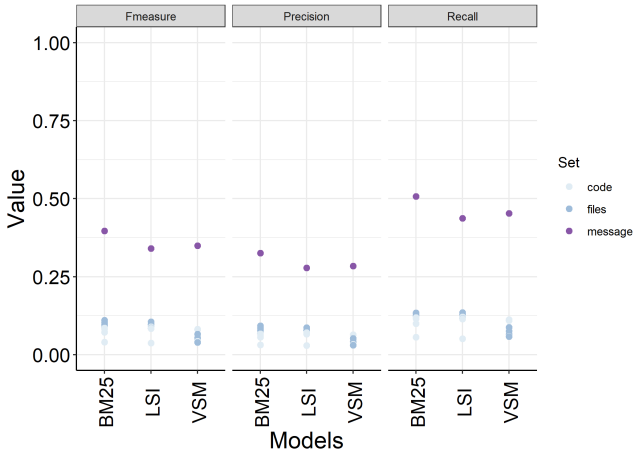

Fig. 4: Effectiveness of the information retrieval models

Figure 5 shows the precision, recall and F-measure of the machine learning models and feature sets. Each model-set box plot limits represent upper and lower quartiles, while the small lines represent the mean. The RF model with the all set achieved the highest performance. The average precision was 93% and recall was 94%. Furthermore, the NN model with all the features obtained a very similar performance, with an average precision of 92% and recall of 94%. Still, both RF and NN had an average of 90% F-measure across the sets. Meanwhile, DT had an average 87% F-measure, thus were slightly less effective than RF and NN. Using all the features was the most effective set and it achieved 93% of precision, recall and F-measure on average. The process set was the second-best set with an average precision of 84% and recall of 98%. Specifically, the recall was very close to or was one for the process set. Lastly, the similarity set had the worst performance achieving an average precision and recall of 83%. Regarding the variance, the all set had a low variance. The all sets variance, on average,

ranged between 1% to 2% for precision and recall. The similarity set variance was slightly more (on average) ranging between 3% to 5%. Lastly, using the process set, the variance was considerable across all models with an average 5% precision variance and 9% recall variance.

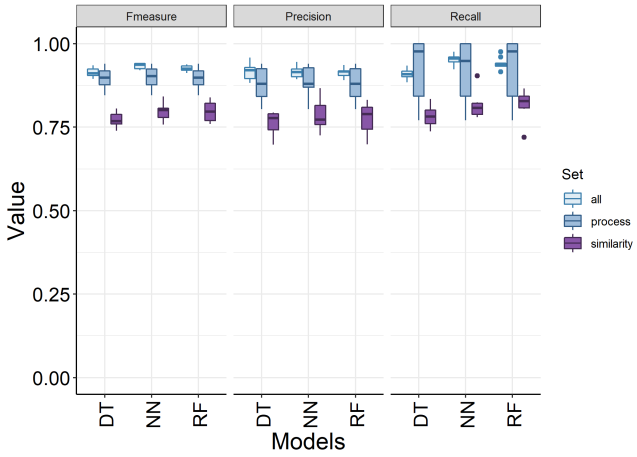

Fig. 5: Effectiveness of the machine learning models

4.2 Discussion

Our findings show that trace links were often incomplete in our project, which provides further empirical evidence of the need to create automatic or semi-automatic trace links. Previous studies reported that manual linking was approximately 48% of commits were not linked to issues, and ranged between 30% and 45% of issues were not linked to commits [10,11,18]. Our student undergraduate project had a similar percentage of missing commits links (43%), though the stories linked were slightly higher (56%) than the average reported by previous studies. We also found evidence that there is inequality in the linking process, with some students and teams students barely contributing to the manual linking. Hence, instructors must give further attention to verify if students are actually linking their commits, possibly with tool support.

We found that information retrieval models could achieve adequate precision by sacrificing the recall value.. This seems to be in line with previous research results for information retrieval techniques in automatic link creation and maintenance, achieving low precision and higher recall [40]. However, these models did provide additional textual information to the machine learning models that helped improve their performance. The machine learning models attained high precision and recall, regenerating most commit-story links (true links) with few false links. Compared to other recent commit-issue approaches that accomplished

and precision within the 90% using machine learning [10], we achieved similar results.

Our results are encouraging as they can be applied in courses where instructors need to retrieve links automatically at any point of the project. With future validation, this approach may be used to help in semi-automatic or automatic link creation. Furthermore, analyzing trace links can provide quantitive information about students' implementation trends during the process. Thus, the data gathered can be used to plan and monitor projects, helping grade assessment and improving the development process.

5 Conclusions

Manually creating trace links is often incomplete; therefore automatic or semi-automatic approaches are needed. Specifically, trace links are helpful in education as they can monitor projects, determine implementation contributions and help grade assessment. In this paper, we automatically recover trace links between commits and user stories in an undergraduate software engineering project. We applied our approach in a student project using agile methodologies with twenty students, 1,035 commits, and 230 committed or done stories. We found evidence that only 43% of the commits were linked to the stories. We evaluated both information retrieval and machine learning models to recover missing links. Algebraic and probabilistic information retrieval models achieved, in the best of the results, 32% precision and 50% recall. Even though information retrieval models did not achieve outstanding performance, they were helpful as they provided textual similarity information of the unstructured data. Thus, machine learning models, using a combination of textual and process data, achieved promising results. In the best case, these models reached 93% of precision and 94% of recall. The results demonstrate the applicability of the approach to recover links in projects to help monitor and evaluation of 'students' contributions during the development of the project.

In future work, we plan to extend our analysis to other undergraduate software engineering projects. Additionally, this work could be further expanded using different models and other data from the commits and issues. Further research is needed in creating automatic trace links in students' projects in-vivo to determine the accuracy and feasibility of the approach to create links. Additionally, creating tools to help monitor the projects' traceability. Further research could also gather students' and instructors' perceptions of the benefits and challenges of traceability in educational projects. Research can also be conducted to gather instructors' and students' benefits and current traceability challenges in an educational context. Finally, students' trends between commits and issues can be analyzed to determine behavioral patterns.

Acknowledgments. This research was partially funded by Universidad de Costa Rica, project No. 834-C1-011.

References

1. J. Cleland-Huang, O. Gotel, and A. Zisman, Eds., *Software and Systems Traceability*. London: Springer London, 2012.
2. H. U. Asuncion, F. François, and R. N. Taylor, "An end-to-end industrial software traceability tool," in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2007, pp. 115–124.
3. P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang, "Strategic traceability for safety-critical projects," *IEEE software*, vol. 30, no. 3, pp. 58–66, 2013.
4. D. Ståhl, K. Hallén, and J. Bosch, "Achieving traceability in large scale continuous integration and delivery deployment, usage and validation of the eiffel framework," *Empirical Software Engineering*, vol. 22, no. 3, pp. 967–995, 2017.
5. P. Hübner and B. Paech, "Interaction-based creation and maintenance of continuously usable trace links between requirements and source code," *Empirical Software Engineering*, vol. 25, no. 5, pp. 4350–4377, Sep. 2020.
6. R. Torkar, T. Gorschek, R. Feldt, M. Svahnberg, U. A. Raja, and K. Kamran, "Requirements traceability: A systematic review and industry case study," *International Journal of Software Engineering and Knowledge Engineering*, vol. 22, no. 03, pp. 385–433, May 2012.
7. R. M. Parizi, S. P. Lee, and M. Dabbagh, "Achievements and Challenges in State-of-the-Art Software Traceability Between Test and Code Artifacts," *IEEE Transactions on Reliability*, vol. 63, no. 4, pp. 913–926, Dec. 2014.
8. S. Winkler and J. von Pilgrim, "A survey of traceability in requirements engineering and model-driven development," *Software & Systems Modeling*, vol. 9, no. 4, pp. 529–565, Sep. 2013.
9. E. Bouillon, P. Mäder, and I. Philippow, "A Survey on Usage Scenarios for Requirements Traceability in Practice," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2013, pp. 158–173.
10. H. Ruan, B. Chen, X. Peng, and W. Zhao, "DeepLink: Recovering issue-commit links based on deep learning," *Journal of Systems and Software*, vol. 158, p. 110406, Dec. 2019.
11. M. Rath, J. Rendall, J. L. C. Guo, J. Cleland-Huang, and P. Mäder, "Traceability in the Wild: Automatically Augmenting Incomplete Trace Links," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, May 2018, pp. 834–845.
12. S. Otte, "Version Control Systems," *Computer Systems and Telematics*, 2009.
13. D. Bertram, A. Voida, S. Greenberg, and R. Walker, "Communication, collaboration, and bugs: The social nature of issue tracking in small, collocated teams," in *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work*, ser. CSCW '10. New York, NY, USA: Association for Computing Machinery, Feb. 2010, pp. 291–300.
14. J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–5, May 2005.
15. S. Hamer, C. Quesada-López, A. Martínez, and M. Jenkins, "Measuring students' contributions in software development projects using Git metrics," in *2020 XLVI Latin American Computing Conference (CLEI)*. IEEE, 2020, p. 10.
16. ——, "Measuring Students' Source Code Quality in Software Development Projects Through Commit-Impact Analysis," in *Information Technology and Systems*, vol. 1331. Cham: Springer International Publishing, 2021, pp. 100–109.

17. J. Cleland-Huang, M. Rahimi, and P. Mäder, "Achieving lightweight trustworthy traceability," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014*. Hong Kong, China: ACM Press, 2014, pp. 849–852.
18. J. Polaczek and J. Sosnowski, "Exploring the software repositories of embedded systems: An industrial experience," *Information and Software Technology*, vol. 131, p. 106489, Mar. 2021.
19. R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "ReLink: Recovering links between bugs and changes," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: Association for Computing Machinery, Sep. 2011, pp. 15–25.
20. A. Nguyen, T. Nguyen, H. Nguyen, and T. Nguyen, "Multi-layered approach for recovering links between bug reports and fixes," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, FSE 2012*, 2012.
21. P. Hübner and B. Paech, "Evaluation of Techniques to Detect Wrong Interaction Based Trace Links," in *Requirements Engineering: Foundation for Software Quality*, E. Kamsties, J. Horkoff, and F. Dalpiaz, Eds. Cham: Springer International Publishing, 2018, vol. 10753, pp. 75–91.
22. Y. Sun, Q. Wang, and Y. Yang, "FRLink: Improving the recovery of missing issue-commit links by revisiting file relevance," *Information and Software Technology*, vol. 84, pp. 33–47, 2017.
23. Y. Sun, C. Chen, Q. Wang, and B. Boehm, "Improving missing issue-commit link recovery using positive and unlabeled data," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Oct. 2017, pp. 147–152.
24. S. Chacon and B. Straub, *Pro Git*. Apress, Nov. 2014.
25. S. Dimitrijević, J. Jovanović, and V. Devedžić, "A comparative study of software tools for user story management," *Information and Software Technology*, vol. 57, pp. 352–368, 2015.
26. M. Cohn, *User Stories Applied: For Agile Software Development*, ser. Addison-Wesley Signature Series. Boston: Addison-Wesley, 2004.
27. R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. ACM press New York, 1999.
28. M. F. Porter, "An algorithm for suffix stripping," *Program*, 1980.
29. S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.
30. M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen, N. Thorat, F. Viégas, M. Wattenberg, G. Corrado, M. Hughes, and J. Dean, "Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 339–351, Dec. 2017.
31. T. Vale and E. S. de Almeida, "Experimenting with information retrieval methods in the recovery of feature-code SPL traces," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1328–1368, Jun. 2019.
32. G. Amati, "Information Retrieval Models," in *Encyclopedia of Database Systems*, L. Liu and M. T. Özsu, Eds. New York, NY: Springer, 2018, pp. 1976–1981.
33. M. Borg, P. Runeson, and A. Ardö, "Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability," vol. 19, no. 6, pp. 1565–1616, Dec. 2014.

34. M. Melucci, "Vector-Space Model," in *Encyclopedia of Database Systems*, L. LIU and M. T. ÖZSU, Eds. Boston, MA: Springer US, 2009, pp. 3259–3263.
35. G. W. Furnas, S. Deerwester, S. T. Dumais, T. K. Landauer, R. A. Harshman, L. A. Streeter, and K. E. Lochbaum, "Information retrieval using a singular value decomposition model of latent semantic structure," in *Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '88. New York, NY, USA: Association for Computing Machinery, May 1988, pp. 465–480.
36. S. Robertson, H. Zaragoza, and M. Taylor, "Simple BM25 extension to multiple weighted fields," in *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, ser. CIKM '04. New York, NY, USA: Association for Computing Machinery, Nov. 2004, pp. 42–49.
37. A. Trotman, A. Puurula, and B. Burgess, "Improvements to BM25 and Language Models Examined," in *Proceedings of the 2014 Australasian Document Computing Symposium*, ser. ADCS '14. New York, NY, USA: Association for Computing Machinery, Nov. 2014, pp. 58–65.
38. R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50.
39. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
40. O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, and G. Antoniol, "The quest for Ubiquity: A roadmap for software and systems traceability research," in *2012 20th IEEE International Requirements Engineering Conference (RE)*, Sep. 2012, pp. 71–80.