2019 IEEE/ACM 10th International Symposium on Software and Systems Traceability (SST)

Comparing Traceability through Information Retrieval, Commits, Interaction Logs, and Tags

Marcus Seiler, Paul Hübner, and Barbara Paech
Heidelberg University, Software Engineering Group, Heidelberg, Germany,
Email: {seiler, huebner, paech}@informatik.uni-heidelberg.de

***Abstract*—*[Context & motivation]* Traceability is used to follow and understand the relationships between various software engineering artifacts such as requirements and source code. Comprehensive traceability of software engineering artifacts is important to ensure that a software can be further developed or maintained. Traceability links are often created manually by using commit ids or retrospectively by using information retrieval (IR). *[Question/Problem]* However, creating traceability links manually is costly and it is error-prone in retrospect. As part of our ongoing research on feature management where traceability is also of interest, we use a lightweight tagging approach to relate artifacts. We want to investigate how such a tag-based approach compares to approaches using commit ids, interaction logs (IL), and IR for creating traceability links. *[Principal ideas/results]* We conducted a case study in which students applied the tag-based, the IL-based, and the commit-based approach. We transformed the tags to traceability links and compared them with the commit-based and IL-based approach as well as with IR-based approaches, using a gold standard. We applied different improvements. *[Contribution]* This is the first study comparing four completely different traceability approaches in one project. The combination of IL and commit ids shows the best precision and recall but is intrusive. The other approaches differ less in precision and recall and both need improvement for practical application. We discuss the benefits and drawbacks of the different approaches and state implications for research and practice.**

I. Introduction

Traceability is used to follow and understand the relationships between various software engineering artifacts such as requirements, design artifacts, or source code. Comprehensive traceability of software engineering artifacts is important to ensure that a software can be further developed or maintained. A recent study shows that traceability has a positive impact on software quality in terms of time to implement a solution and the correctness of the solution [1]. Although traceability has a positive effect, it is still not implemented thoroughly in current development practices. There are many ways to create traceability links. Using a commit-based link creation approach is common practice in open source projects [2]. Many link creation approaches are based on information retrieval (IR), but recovering links automatically often leads to incomplete and incorrect links [3]. Interaction logging (IL) approaches focus on increasing precision and thus reduce incorrect links [4]. As part of our ongoing research on feature management where traceability is also of interest, we use a lightweight tagging approach to relate artifacts [5].

We want to investigate how such a tag-based approach compares to commit-, IL-, and IR-based approaches for creating traceability links. In this paper, we report on a case study in which students applied a commit-, IL-, and the tag-based approach for relating requirements and code. We also applied IR-based approaches retrospectively. We compared the links created by the four different approaches regarding precision and recall using a gold standard. We also applied different improvements to the approaches. Moreover, we compare the approaches with a special focus on our feature tagging approach considering the size of features and the effort to create the links.

This study extends the results of [4] by using a slightly different data set, an additional traceability approach (the tags) and by providing a more thorough discussion.

Our results show that all approaches can achieve good to excellent precision and all approaches besides the commit-based approach achieve acceptable to excellent recall. The combination of IL and commit ids outperforms the other approaches in every aspect. However, the interaction logging is intrusive. With the focus on features, tags show good precision for smaller features and good recall for bigger ones. Tags need relatively little effort. It is an open question whether the coarser-grained traceability provided by tags is sufficient for practical application.

The remaining paper is structured as follows: Section II introduces the fundamentals to evaluate traceability links and presents the different link creation approaches. Section III discusses related work. Section IV describes the case study with its context, research questions, and hypotheses. Section V presents the results of the study and Section VI discusses them. Section VII discusses threats to our study and Section VIII finally concludes this paper.

II. Background

This section briefly introduces the fundamentals of trace link evaluation. Then, it describes the four different link creation approaches used in our study.

A. Evaluation of Traceability Links

Precision (P) and recall (R) are typically used to measure and compare the quality of link creation approaches [6], [7]. According to [4], P is the amount of correct links (true positives, TP) of all links found by an approach, i.e., the sum of TP and wrong links (false positives, FP). R is the amount

2157-2194/19/$31.00 ©2019 IEEE
DOI 10.1109/SST.2019.00015

of TP links found by an approach of all existing correct links, i.e., the sum of TP and false negative (FN) links.

$$P = \frac{TP}{TP + FP} \qquad R = \frac{TP}{TP + FN} \qquad F_1 = 2 \cdot \frac{P \cdot R}{P + R}$$

The F_1 score combines precision and recall in a single measurement to compare the accuracy of the link creation approaches. F_1 weighs the results for precision and recall equally and thus does not favor one or another. We show F_1 and discuss precision and recall in detail.

B. Approaches to Create Traceability Links

We describe the four different trace link creation approaches in the following.

1) IR-based Link Creation: IR searches for information within a set of artifacts and aims to retrieve all relevant artifacts while minimizing the non-relevant artifacts [8]. IR is useful to link requirements to code files [9]. IR typically uses the cosine similarity to measure the similarity between artifacts, which is a numerical value between 0 and 1 [10]. 0 indicates no similarity between two artifacts and 1 indicates that two artifacts are identical. A threshold value for the similarity is used to define whether two artifacts are related with each other and should therefore be linked [11]. Thus, varying this threshold also varies the number of created links.

There are different IR algorithms. The most common IR algorithms used for automatic trace link creation are the vector space model (VSM) and latent semantic indexing (LSI) [10]. VSM uses a more strict term comparison than LSI. Whereas VSM measures the similarity based on terms only, LSI measures the similarity also considering synonyms [8]. Thus, LSI enables similarity matches between artifacts, which do not contain exactly the same terms.

2) Commit-based Link Creation: Commit-based trace link creation approaches relate the code committed to version control systems to other artifacts such as requirements. A common commit-based approach in issue tracking systems (ITS) is to provide the id of an issue describing a requirement or a development task within the commit message [2]. Thus, a trace link between the issue and the code in the commit is created based on the issue id at commit time.

3) IL-based Link Creation: IL-based approaches use interaction logging to create trace links. The approach by Hübner and Paech [4] uses both, recorded interactions and commit ids for link creation. The IL-based approach works as follows: First, interaction events in the development environment of a developer are recorded until the developer performs a commit with a commit message, which contains an issue id. The recorded interactions are mapped to this issue. Second, the approach generates trace links between the requirements related to the issue and the code files affected by the interactions.

4) Tag-based Link Creation: Tagging is an effective approach to attach additional information to artifacts [12]. We developed a lightweight tool to manage feature knowledge in ITS [5]. Feature knowledge comprises feature descriptions and all related software engineering artifacts such as requirements, work items, and code, as well as their relations. Instead of creating traces between feature knowledge, feature knowledge is tagged with the same keyword. In particular, we use tags for feature descriptions, requirements, work items, and code.

The tag-based approach works as follows: One tag for each feature of a software is used. The tag summarizes the feature in a short and concise manner with a descriptive term. Tags are manually defined during development, prior to their application. Then, developers apply these tags to feature knowledge. This tagging method adheres to the following rules: (i) A feature description is tagged with a feature tag if and only if it contains the description of the feature. (ii) A requirement is tagged with a feature tag if and only if the requirement refines the feature. (iii) A work item is tagged with a feature tag if and only if the described task addresses specification, quality assurance, or implementation of the feature. (iv) Source code is tagged with a feature tag if and only if the source code implements (parts of) the feature.

For this study, we transformed the tags to trace links as follows: We created a trace link between a requirement and a code file if both the requirement and the code file have the same tag applied. We took care of possible duplicate links resulting from multiple tags applied to the same requirements and code files. For example, if requirement R and code file C were both tagged with tags T_1 and T_2, we only created one trace link between R and C.

III. Related Work

We discuss related work for the four approaches regarding precision and recall. The values for precision and recall have been obtained with different data sets. Therefore, these values are specific to the data set used. The reported values serve as an example for the precision and recall spectrum but are not directly comparable.

De Lucia et al. [13] evaluated LSI. They report values of 90% for recall and 25% for precision for structured requirements and source code. Ali et al. [14] studied VSM in three projects. They report results for precision of 15% with a recall of 16% and a precision of 72% with a recall of 7%. Merten et al. [15] compared different IR-based approaches for unstructured requirements and report precision values up to 11%, while optimizing recall up to 100%. Thus, precision of IR was worse for unstructured requirements than for structured requirements.

Regarding the commit-based approach, Rath et al. [2] report that only 60% of the commits contain an issue id. Thus, values of up to 60% can be expected for recall. To mitigate missing ids and thus to increase the recall, they investigated the use of machine learning to identify matching issues for commits without ids. They report an average recall of 91.6% with a precision of 17.3% for their trained classifiers.

Hübner and Paech [4] evaluated their IL-based approach. They report values ranging from 85% up to 90% for precision and values ranging from 67% up to 79% for recall. Also, they compared their approach with a commit-based approach on the same data set. They report precision values ranging from

62% to 67% and recall values ranging from 42% to 47% for the commit-based approach.

Hale et al. [16] analyzed the usage of tags for linking brainstorming notes to development artifacts. They report that 25% of the used tags create links between notes and artifacts, and that 42% of the tagged notes relate to artifacts. However, the study does not provide values for precision and recall. Also, they did not complete the comparison of their tag-based approach and trace links in this preliminary study.

There are also other approaches to create traceability links besides the chosen four approaches. For example, Guo et al. [17] present an approach that uses deep learning to create links between requirements. They experimented with different settings and also compared their approach with VSM and LSI. Their results show that the use of deep learning increases both the precision and the recall and in general performs better than VSM and LSI. The best precision of the approach is about 27% with a recall of about 8%. In comparison, VSM has a best precision of 25% with a recall of about 3% and LSI has a best precision of 25% with a recall of about 15% for the same data set. Also, the approach showed better precision and recall with increasing training data. The best precision is about 38% with a recall of about 9% for the larger training set. During our study, this approach was not available.

IV. Case Study Design

We conducted a study with students in order to evaluate the four approaches. It is the same study as used by Hübner and Paech in [4]. The tag-based approach was not evaluated by their study. We restricted the data set for reasons explained below.

In the following, we describe the context of the case study, the research questions and the hypotheses in Section IV-A and Section IV-B, respectively. We describe the data sources in Section IV-C and the creation of the gold standard in Section IV-D. We describe the tool support and the set-up of IR in Section IV-E, and the improvements for the approaches in Section IV-F.

A. Study Context

The study was performed during a development project with six students over a period of six months. The project lasted from October 2017 to March 2018. The students developed an indoor navigation app for Android-based devices for a real customer. Primary users are (other) students who use the app to locate and navigate rooms where lectures take place. Also, the app is able to retrieve information from public transportation allowing to display the departure schedule of a nearby station. The customer was a company developing mobile apps. The development method was Scrum-like. In each sprint, one of the students acted as Scrum master and thus was responsible for development planning and communicating with the customer. The customer provided a high-level vision description of the app. The students derived features and refined them during development in agreement with the customer. They used Jira with epics to describe features, with user stories to refine features, and with work items to describe development tasks. The students used Git for Java source code and Eclipse as development environment. They applied the commit-, tag-, and IL-based approach, and used plug-ins for Jira and Eclipse to support the latter two approaches. At the beginning of the project, the commit-, tag-, and IL-based approaches, and the basic usage of the plug-ins were introduced. Also, the students were supported in the initial set-up of the plug-ins.

B. Research Questions

The goal of this study is to compare different trace link creation approaches. For this, we first want to compare the four approaches with respect to precision and recall in general. As our tag-based approach focuses on relating features with a requirement and code rather than relating a requirement with code directly, we want to investigate if precision and recall differ for different sizes of features. We employ the same improvements as applied by Hübner and Paech [4] to investigate whether such improvements have an effect on precision and recall. Finally, we want to compare the effort for creating links using the different approaches, as lightweight approaches are important for practical application. We therefore raise the following research questions and hypotheses:

RQ_1: How do the approaches compare for precision and recall? For IR, IL, and commit, we expect similar values for precision and recall for unstructured requirements as reported by others (cf. Section III). Due to the many-to-many nature of the tags, which results in many false positives, decreasing the precision, we expect a worse precision for tags compared to the other approaches. We hypothesize that the IL-based approach outperforms the other approaches regarding the precision, as the IL-approach focuses on precision optimization.

RQ_2: How do the approaches compare for precision and recall regarding the size of features? We hypothesize better precision and recall for smaller features than for bigger ones across all approaches. The functionality of smaller features is more distinguishable in the code, making link creation easier.

RQ_3: How do the approaches compare for precision and recall after improvements? We hypothesize that the improvements increase the precision and recall for all approaches.

RQ_4: How do the approaches compare for the effort to create the links? We hypothesize that the tag-based approach requires less effort than the IL- and the commit-based approach.

C. Data Sources

We used four different data sources for our evaluation. The data of the first three sources are subsets of the data used in [4]. We used the features and feature tags applied to requirements and code files as an additional data source.

1) Source code under version control: The repository comprised 406 commits from which 226 commits contained an issue id. Build configurations, readmes, binaries, and 3rd party

library files were excluded to focus on code created by the developers. The first 395 commits were used for link creation, as the remaining 11 commits were performed after the official end of the project and these commits did not contain an issue id. The proportion (57.22%) of issue ids in commits is similar to others [2]. The repository contained 40 java and 26 xml files. Our tag-based approach works only with java files, as we use the built-in java annotations for code tags. Therefore, in contrast to [4] we excluded the 26 xml files. 38 of 40 code files are linked to an issue using a commit id.

2) Requirements in ITS: The ITS contained 23 stories in total. Three stories did not describe requirements, but described testing and project organization. The state of three other stories was unresolved at the end of the project. We excluded the unresolved stories as well as the testing and organizational stories. We finally used 17 requirements and their 74 sub-tasks for evaluation. These are the same requirements as used by [4].

3) Interaction Recordings: The interaction recordings for the 17 stories and 74 sub-tasks comprise 6471 interaction events. We used 2709 interaction events to create trace links while Hübner and Paech [4] used 4012 interaction events. The interaction events comprise a time stamp, the file involved in the interaction, the type of interaction (edit or select), the frequency as the number of all interactions performed for a file, and the part of the development environment in which the interaction was performed. The remaining 3762 interaction events were removed, as these interactions were out-of-scope, e.g. interaction events for excluded files.

4) Features and Feature Tags: The ITS contains five different tags for the five features of the developed software. These five feature tags were defined upfront (prior to applying the tags to issues and code) and in agreement with the developers and the customer. Each feature tag summarizes the feature in a short and concise manner, e.g. the tag *RouteFinding* was used to summarize the feature for searching points on a map, finding a route between two points, and navigating from one point to another inside the app.

Table I shows the number of stories and code files per feature. We quantified features as big or small using the amount of stories refining a feature and the amount of code files implementing parts of a feature. Thus, it indicates the effort for the developers to realize the features. We consider a feature as big if it comprises many requirements and is also implemented in many code files. The *RouteFinding* feature comprises 8 requirements and 27 code files, and is therefore the big feature in the project. The other four features *Filter, PublicTransportation, Tweets* and *CourseCatalog* together comprise the remaining 9 requirements and are implemented in 14 code files. These are the small features.

The students applied the feature tags manually to the stories. Although we provided a tag recommendation (cf. Section IV-E), the students used them in 14% of the stories only. Each of the 17 stories had exactly one of the tags applied. One tag was applied to a story, but was never applied to one of the code files.

TABLE I
NUMBER OF STORIES AND CODE FILES PER FEATURE

Feature	*# Stories*	*# Code files*
RouteFinding	8	27
Filter	4	4
PublicTransportation	3	11
Tweets	1	4
CourseCatalog	1	0

TABLE II
DISTRIBUTION OF FEATURE TAGS IN THE CODE FILES

# Applied feature tags	*# Code files*
0	3
1	33
2	1
3	1
4	2

The students applied feature tags manually to the code. Table II shows the distribution of tags for the code files. Three code files contain no tags. The majority of 33 code files contain one tag. Two and three tags are each contained in one code file. Two code files contain four tags. Overall, 37 of 40 code files had at least one of the tags applied. Each code file contains 1.15 tags on average with a maximum of four and a minimum of zero. In our study, we neither excluded untagged code files nor the story whose tag did not occur in code.

D. Gold Standard

We used the gold standard created by Hübner and Paech [4]. The six developers of the project created the gold standard. The developers vetted link candidates between requirements and the source code.

In contrast to [4], we removed all non-java files and their links, as our tag-based approach only works for java files. The gold standard comprises 189 trace links with 17 requirements linked to 40 java files. In comparison, the original gold standard contained 309 trace links for 17 requirements and 66 code files.

E. Tool Support and IR Set-up

The students used tool support for the IL- and the tag-based approach. The tool support for the IL-based approach comprises an Eclipse plug-in, which automatically captures interaction events. The captured interaction events are automatically bundled and uploaded to the issue specified by its id within a commit message. The students had no direct contact with the captured interactions nor did they use them for any other purpose.

The tool support for the tag-based approach comprises a plug-in for Jira and a plug-in for Eclipse. The Jira plug-in provides a recommendation engine to suggest feature tags for issues by processing the issue description and presenting the most likely feature tag for an issue to the students. In addition, the plug-in automatically inherits feature tags applied to requirements to corresponding sub-task issues (work items) refining a requirement. The Eclipse plug-in completes a feature

tag in the code when typing parts of the tag name. Only feature tags, which have been applied to the code previously are completed.

We applied IR in retrospect. We experimented with different settings of VSM and LSI prior to the final evaluation. We performed the same preprocessing as in [4]. Thus, we removed stop words and punctuation, performed character removal and stemming. The camel case notation was used in the code. Therefore, we also performed camel case identifier splitting [13]. The threshold 0.2 for VSM and the threshold 0.03 for LSI showed the best F_1 scores in our preliminary tests. Thus, we used these thresholds for the evaluation.

F. Improvements

Hübner and Paech [4] used precision and recall improvements, which we also employ for our evaluation.

First, they improve the precision for the IL-based approach by removing potentially wrong links using the interaction-specific meta-data frequency, duration, and event type. Clearly, this optimization cannot be used to improve the precision of the other approaches, as this is specific to the IL-based approach.

Second, they utilize the code structure of code files to improve the precision and the recall for all approaches. The code structure denotes relations between code files such as references. They found that precision can be improved if only those code files are linked to a user story which are linked to at least one other code file of the same user story. The code structure of code files which are already linked to a user story is utilized to improve the recall. A link between a code file and a user story is added, if the code file is referenced by another code file that is already linked to the user story.

V. Results

We present the results for our research questions in the following section. We use the following notation for the different approaches:

- FT denotes the tag-based approach.
- $ComL$ denotes the commit-based approach.
- IL_{com} denotes the IL-based approach.
- $IR(LSI_{0.03})$ denotes the IR-based approach using LSI with a threshold of 0.03.
- $IR(VSM_{0.2})$ denotes the IR-based approach using VSM with a threshold of 0.2.
- X_i denotes an approach with improvements where $X \in \{FT, ComL, IL_{com}, IR(LSI_{0.03}), IR(VSM_{0.2})\}$, e.g., FT_i denotes the tag-based approach with improvements.

Table III shows the results for RQ_1, RQ_2, and RQ_3.

A. Results for RQ_1: Comparison of Precision and Recall

The results for RQ_1 are shown in the corresponding part of Table III. FT has a precision of 44.64% and a recall of 63.49% resulting in a F_1 of 52.40%. $ComL$ has a precision of 61.64% and a recall of 51.85% resulting in a F_1 of 56.32% Both $IR(LSI_{0.03})$ and $IR(VSM_{0.2})$ show the best results for recall (74.60-83.60%) and the worst for precision (39.72-37.62%) and a similar F_1 around 51%. IL_{Com} provides the best result for precision (83.75%) with a recall of 70.90%, resulting in the best F_1 with 76.79%.

Overall, our evaluation shows similar results for the approaches as in other studies. We have to reject our hypothesis partly for the tag-based approach. The precision is worse compared to the commit- and IL-based approach, but better than the IR-based approach. Moreover, the F_1 scores do not substantially differ for the tag-, commit-, and IR-based approaches. The results confirm our hypothesis for the IL-based approach.

B. Results for RQ_2: Comparison of Precision and Recall for Feature Size

The results regarding the size of features (RQ_2) are shown in the corresponding parts of Table III. Precision and recall for $IL_{Com}, IR(VSM_{0.2})$ and $IR(LSI_{0.03})$ do not substantially differ for small or big features and thus to the results of RQ_1. The recall of $ComL$ is nearly identical for the small and big features, but the precision for smaller features (68.06%) is better than for the big feature (55.06%). Thus, compared to RQ_1 precision for small features is much improved, but leads to worse recall and the other way round for big features. FT provides a precision of 75.47% for small features with a recall of 41.67% and a precision of 37.04% with a recall of 86.02% for the big feature. Compared to the other approaches, FT provides the best recall value for the big feature. For small features, still the IR-approaches provide the best recall. In both cases, still IL_{Com} has the best precision.

Our hypothesis holds for IR. As the recall of FT and $ComL$ is worse for smaller features, we have to reject our hypothesis for FT and $ComL$.

C. Results for RQ_3: Comparison of Precision and Recall for Improvements

The results regarding RQ_3 are shown in the corresponding part of Table III. The precision of IL_{Com} and $ComL$ could be improved to 91.16% and 63.4%, respectively. In the case of FT and IR, the improvements lead to a worse precision, but the degradation is less pronounced in IR than for FT. The recall could be improved for all approaches. Except for IL_{Com}, where the recall has been improved by 16%, the recall for the other approaches has been improved by only 5%.

The improvements increase both precision and recall for IL_{com} and $ComL$. Our hypothesis holds for IL_{com} and $ComL$, but not for FT and IR.

D. Results for RQ_4: Comparison of the Effort for Link Creation

We compare the effort to create trace links using the approaches from the developers' point of view. Here, we do not include manual improvements through vetting of links. This is discussed in Section VI-B.

The gold standard contains 189 links between 17 requirements and 40 code files. As described in Section IV-C1, 38 of

TABLE III
RESULTS FOR PRECISION AND RECALL OF THE FOUR LINK CREATION APPROACHES

Approach	*# GS links*	*# links*	*# TP*	*# FP*	*# FN*	*Precision*	*Recall*	F_1
Pure (RQ_1)								
FT	189	269	120	149	69	44.61%	63.49%	52.40%
$ComL$	189	159	98	61	91	61.64%	51.85%	56.32%
IL_{Com}	189	160	134	26	55	**83.75%**	**70.90%**	**76.79%**
$IR(LSI_{0.03})$	189	355	141	214	48	39.72%	74.60%	51.84%
$IR(VSM_{0.2})$	189	420	158	262	31	37.62%	83.60%	51.89%
Small features only (RQ_2)								
FT	96	53	40	13	56	75.47%	41.67%	53.69%
$ComL$	96	72	49	23	47	68.06%	51.04%	58.33%
IL_{Com}	96	82	68	14	28	**82.93%**	70.83%	76.40%
$IR(LSI_{0.03})$	96	193	77	116	19	39.90%	80.21%	53.29%
$IR(VSM_{0.2})$	96	214	83	131	13	38.79%	**86.46%**	53.55%
Big features only (RQ_2)								
FT	93	216	80	136	13	37.04%	**86.02%**	51.78%
$ComL$	93	89	49	40	44	55.06%	52.69%	53.85%
IL_{Com}	93	78	66	12	27	**84.62%**	70.97%	77.19%
$IR(LSI_{0.03})$	93	162	64	98	29	39.51%	68.82%	50.20%
$IR(VSM_{0.2})$	93	208	75	133	18	36.06%	80.65%	49.83%
With Improvements (RQ_3)								
FT_i	189	317	131	186	58	41.32%	69.31%	51.78%
$ComL_i$	189	167	106	61	83	63.47%	56.08%	59.55%
IL_{Com_i}	189	181	165	16	24	**91.16%**	**87.30%**	**89.19%**
$IR_i(LSI_{0.03})$	189	392	153	239	36	39.03%	80.95%	52.67%
$IR_i(VSM_{0.2})$	189	441	163	278	26	36.96%	86.24%	51.75%

the 40 code files were linked with 226 commits. Thus, each of these commits must contain at least one id. Therefore, the developers had to provide at least one id for 226 commits to create the links between requirements and code files in $ComL$. Developers have no additional effort for recording interactions in IL_{Com}, but they have the same effort to provide the issue id as for $ComL$. Thus, developers have the same effort for creating links using IL_{Com} and $ComL$. In FT, the developers need to apply tags for 17 requirements and 40 code files leading to 57 tag applications. In our view, the mental model of the developer for choosing the correct id at commit time and choosing the correct feature tag is the same. The effort to apply FT is therefore low compared to the IL_{Com} and $ComL$, but clearly higher than for IR, which requires no effort from the developers. Thus, our hypothesis for RQ_4 holds.

VI. DISCUSSION

The discussion of the results has three parts. First, we discuss our results in direct comparison to the results of Hübner and Paech [4] in Section VI-A. Then, we discuss the results in general in Section VI-B. Finally, we present implications for research and practice in Section VI-C.

A. Study Comparison

For comparison, Table IV shows the results for the three approaches $IL_{Com}, ComL$, and IR on the extended data set presented in [4].

Overall, there are only minor differences for the F_1 scores compared to our study. The precision of IL_{Com} and $ComL$, but not of IR is slightly better than in our study. However, the recall values of all approaches in our study are much higher. Thus, it seems easier for developers to provide commit ids for java files than for xml files. Also, the textual similarity of requirements to java files seems to be stronger.

TABLE IV
RESULTS FOR $ComL, ComL_i, IR, IR_i$ AND IL_{Com} [4]

Approach	*Precision*	*Recall*	F_1
$ComL$	66.8%	41.7%	51.4%
$ComL_i$	67.5%	44.3%	53.5%
IL_{Com}	84.9%	67.3%	75.1%
IL_{Com_i}	90.0%	79.0%	84.1%
IR	33.5%	49.2%	39.8%
IR_i	36.9%	55.7%	44.4%

In their evaluation, the applied improvements led to better recall and precision values for each of the approaches. This is similar for the IL- and commit-based approaches in our study. IR has a better recall, but for LSI a slightly worse precision. The same holds for FT. As recall improvements relied on the code structure, also the code structure between java files seems to be slightly different from the code structure between and to xml files.

We therefore conclude that in our study the type of code file has a direct influence on the precision and recall of the link creation approaches.

B. General Discussion

We had to reject the hypothesis for RQ_2 that FT and $ComL$ perform better for smaller features than for bigger ones. This seems to show that for developers providing additional information is not easier for smaller features than for bigger features. However, we distinguished small and big features only quantitatively. We have to investigate other attributes to determine the size of features such as the relation of a feature to other features [18].

TABLE V
BEST RESULTS FOR PRECISION AND RECALL OF THE FOUR APPROACHES

Approach	*Precision*	*Recall*	F_1
$ComL_i$	63.47%	56.08%	59.55%
IL_{Com_i}	91.16%	87.30%	89.19%
FT	44.61%	63.49%	52.40%
$IR_i(LSI_{0.03})$	39.03%	80.95%	52.67%
$IR(VSM_{0.2})$	37.62%	83.60%	51.75%

We also had to reject the hypothesis that the precision improvements work for FT and IR. The latter is in contrast to [4], but cannot be attributed to the code types as the improvements worked for the IL-approach. We therefore have to further investigate the reasons for this as part of future work.

With respect to the effort (RQ_4), IR and FT are better compared to $ComL$ and IL_{Com}. However, clearly the achieved precision and recall values have to be taken into account.

Table V shows the best values for precision and recall.

The precision of IR-based approaches is poor compared to the other approaches. Interestingly, the IR-based approaches show very good recall results for unstructured requirements. However, due to the worse precision which requires a lot of effort for vetting, IR-based approaches are not lightweight. Also, improvements to recall often decrease the precision of IR-based approaches dramatically as shown by Merten et al. [15].

The commit-based approach shows a good precision value, but a poorer recall. As described in Section IV-C1, in our study as in other contexts only about 57% of the commits have an id. Therefore, the goal should be to maximize the recall of $ComL$. As described in Section III, machine learning can be applied to increase the recall of the commit-based approach. Another direction could be to recommend suitable ids for to be committed code [19].

The tag-based approach shows a slightly better precision compared to the IR-approaches, but a worse recall. Due to the nature of tagging, we cannot expect a precision significantly better than 50% for FT. The main goal for FT should be to maximize the recall while keeping the manual effort as low as possible. As shown in the results for the improvements, the code structure improvements did not help. Instead, the recall of the tags itself should be maximized. As for the commit-based approach, the effort to apply tags can be reduced by using recommendations. Such a system can also be used for recall improvement, as additional feature tags for untagged or incompletely tagged code can be suggested. An automatic tag application for untagged or incompletely tagged code files could increase recall without adding additional effort for developers. However, automatic application could result in many false positives, decreasing precision.

IL_{Com} outperforms the other approaches in every aspect. However, not every developer might feel comfortable that their interactions (even if only within the development environment) are permanently recorded and evaluated. Also the effort to apply IL_{Com} is the same as for applying $ComL$.

C. Implications for Research and Practice

Clearly, the results of our study are specific to the used data set and strongly depend on the setting (e.g. chosen preprocessing and thresholds for IR). The comparison of several approaches on two similar data sets showed that even small changes in the research setting have effects on precision and recall. We have seen that using a slightly different data set (in our case using only a subset of code files) had a positive influence on the results of the IR-based approaches. Moreover, we have seen that improvements made in one study cannot always be fully replicated in another even if a nearly identical data set is used. It is therefore important that researchers clearly describe the properties of their data sets. Furthermore, they should build suitable subsets of their data and investigate the impact on precision and recall. Based on our study, it seems interesting to investigate whether some approaches perform better on pure java files.

Our study also showed that there is no best choice for application in practice. For some domains, where tracing is required (e.g. for safety-critical software), the intrusiveness of the IL-based approach and the involved effort might be acceptable. For other domains, little effort is more important and practitioners could benefit from using a tag-based approach as it is more lightweight compared to the commonly used commit-based approach. Moreover, the feature tags enrich the code with semantics and thus could help to better understand the intention behind a piece of code [20].

An open question is whether practitioners really need fine-grained traces. We therefore want to investigate whether the coarser-grained traceability provided by tags is sufficient for practical application.

VII. THREATS TO VALIDITY

Runeson et al. [21] suggest discussing the four threats for empirical studies: construct validity, internal validity, external validity, and reliability.

As this study can be treated as a kind of replication of the study provided by Hübner and Paech [4], their reported threats also apply to this study.

A. Construct Validity

As described in [4], the preprocessing steps and the thresholds have a direct influence on the quality of IR-based approaches and thus influence the comparison with other approaches. The threat is mitigated by the fact that, in this study we applied the same preprocessing steps and same threshold for VSM as in [4]. We chose the threshold of 0.03 for LSI as it provided the best F_1 score in preliminary tests.

B. Internal Validity

A substantial source of bias is that the students vetted the links in the gold standard as in [4]. However, this ensured that the experts created the gold standard. The students knew that the researchers had developed the IL- and tag-based approaches and their tool support. The threat was mitigated as the researcher appreciated both positive and negative feedback

from the students during the use of the approaches. An introduction to the approaches and the tools ensured the same knowledge across all students. The motivation of the students to apply the approaches correctly might be influenced by worries about grades. However, the researchers were not involved in the final grading and the correct application of the approaches had no influence on the grades.

C. External Validity

The results are specific to this development project, e.g., the availability of interaction logs and tool support. The reported results can be different for other settings, e.g., for other IR-approaches. During the project, the students were advised to use the tag- and IL-based approach. The students may behave differently if the usage is not mandatory. Moreover, the size of the development project is limited. The results for larger projects can be different from the results reported in this study. Thus, the generalizability is clearly limited.

D. Reliability

Other researchers might interpret the results in another direction. The researchers documented the steps during design, data collection, and analysis. In addition, another researcher reviewed the design of the case study and the steps for analysis to increase reliability. This also ensures the reproducibility of the study.

VIII. Conclusion & Future Work

In this paper, we investigated the quality of traceability links created by four completely different approaches. We conducted a case study in which students applied a commit-, a tag-, and an IL-based approach to create traceability links. We applied IR-based approaches retrospectively. We compared the quality of all approaches with respect to precision and recall. The results show that the IL-based approach provides the best precision and recall values. The comparison with [4] revealed that the type of the code files has an influence on precision and recall values and the effectiveness of improvements. The effort to apply the tag-based approach is less than compared to a commit- or IL-based approach.

In future work, we therefore want to investigate whether the coarser-grained traceablity provided by tags is sufficient in practice. We also want to investigate recall improvements for the tag-based approach. Also of interest is to investigate whether the effort to apply the commit-, IL-, and tag-based approach could be minimized.

Acknowledgements

We would like to thank all students for their effort in this study.

References

[1] P. Mäder and A. Egyed, "Do developers benefit from requirements traceability when evolving and maintaining a software system?" *Empirical Software Engineering*, vol. 20, no. 2, pp. 413–441, 2014.

[2] M. Rath, J. Rendall, J. L. C. Guo, J. Cleland-Huang, and P. Mäder, "Traceability in the wild: Automatically augmenting incomplete trace links," *CoRR*, vol. abs/1804.02433, 2018.

[3] 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," *20th IEEE International Requirements Engineering Conference (RE)*, pp. 71–80, 2012.

[4] P. Hübner and B. Paech, "Increasing Precision of Automatically Generated Trace Links," in *25th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, 2019, accepted to appear.

[5] M. Seiler and B. Paech, "Using tags to support feature management across issue tracking systems and version control systems," in *23rd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, 2017, pp. 174–180.

[6] J. H. Hayes, G. Antoniol, and Y. Guéhéneuc, "Prereqir: Recovering pre-requirements via cluster analysis," in *15th Working Conference on Reverse Engineering*, 2008, pp. 165–174.

[7] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery," in *27th International Conference on Software Maintenance (ICSM)*, 2011, pp. 133–142.

[8] A. Singhal, "Modern Information Retrieval: A Brief Overview," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 24, no. 4, pp. 35–43, 2001.

[9] A. De Lucia, M. D. Penta, and R. Oliveto, "Improving source code lexicon via traceability and information retrieval," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 205–227, March 2011.

[10] M. Borg, P. Runeson, and A. Ardö, "Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability," *Empirical Software Engineering*, vol. 19, no. 6, pp. 1565–1616, dec 2014.

[11] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova, "Best Practices for Automated Traceability," *Computer*, vol. 40, no. 6, pp. 27–35, jun 2007.

[12] M.-A. Storey, J. Ryall, J. Singer, D. Myers, L.-T. Cheng, and M. Muller, "How Software Developers Use Tagging to Support Reminding and Refinding," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 470–483, 2009.

[13] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering Traceability Links in Software Artifact Management Systems using Information Retrieval Methods," *ACM Transactions on Software Engineering and Methodology*, vol. 16, no. 4, pp. 1–50, 2007.

[14] N. Ali, Y.-G. Guéhéneuc, and G. Antoniol, "Trustrace: Mining Software Repositories to Improve the Accuracy of Requirement Traceability Links," *IEEE Transactions on Software Engineering*, vol. 39, no. 5, pp. 725–741, may 2013.

[15] T. Merten, D. Krämer, B. Mager, P. Schell, S. Bürsner, and B. Paech, "Do Information Retrieval Algorithms for Automated Traceability Perform Effectively on Issue Tracking System Data?" in *22nd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, 2016, pp. 45–62.

[16] M. Hale, N. Jorgenson, and R. Gamble, "Analyzing the role of tags as lightweight traceability links," in *6th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, 2011, pp. 71–74.

[17] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques," in *39th International Conference on Software Engineering (ICSE)*, 2017, pp. 3–14.

[18] A. Classen, P. Heymans, and P.-y. Schobbens, "What ' s in a Feature: A Requirements Engineering Perspective," in *11th International Conference on Fundamental Approaches to Software Engineering (FASE)*, 2008, pp. 16–30.

[19] M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, *Recommendation Systems in Software Engineering*. Springer Publishing Company, Incorporated, 2014.

[20] M. Sulír, M. Nosáľ, and J. Porubän, "Recording concerns in source code using annotations," *Computer Languages, Systems & Structures*, vol. 46, pp. 44–65, nov 2016.

[21] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering: Guidelines and Examples*, 1st ed. John Wiley & Sons, 2012.