Increasing Precision of Automatically Generated Trace Links

Paul Hübner(✉) and Barbara Paech

Institute for Computer Science, Heidelberg University,
Im Neuenheimer Feld 205, 69120 Heidelberg, Germany
{huebner,paech}@informatik.uni-heidelberg.de

Abstract. [**Context and Motivation**] In order to use automatically created trace links during a project directly, the precision of the links is essential. Our interaction-based trace link creation approach (IL) utilizes the interactions of developers recorded in an integrated development environment (IDE) while working on a requirement. For this, developers need to indicate the requirement they are going to work on before coding. This approach worked well in an open-source project with developers who were interested in the interaction logs, but did not work well with students who were not particularly motivated to trigger the interaction recording. [**Question/problem**] Developers often create trace links themselves by providing issue identifiers (IDs) in commit messages. This causes little effort and does not require the awareness for interaction recording. However, as confirmed by recent research, typically only 60% of the commits are linked. In this paper, we study whether and how IL can be improved by a combination with links created by issue IDs in commit messages. [**Principal ideas/results**] We changed our approach so that interaction logs are associated with requirements based on the IDs in the commit-messages. Thus, developers do not need to manually associate requirements and interaction logs. We performed a new student study with this approach. [**Contribution**] In this new study, we show that with this new approach and link improvement techniques precision is above 90% and recall is almost 80%. We also show that for our data this is better than using commit-messages only and better than the often used information retrieval-based approaches.

Keywords: Traceability · Interaction · Requirement · Source code · Precision

1 Introduction

Existing trace link creation approaches are most often based on information retrieval (IR) and on structured requirements, such as use cases [3,5]. These approaches mostly focus on the optimization of recall [3]. In addition, their precision is bad which makes the approaches not applicable when directly using created links [5]. Therefore, a review of the created link candidates by an expert

E. Knauss and M. Goedicke (Eds.): REFSQ 2019, LNCS 11412, pp. 73–89, 2019.
https://doi.org/10.1007/978-3-030-15538-4_5

is necessary before their usage. In security-critical domains such as aeronautics and the automotive industry, complete link sets are required. These links are only created periodically, when needed for certification to justify the safe operation of a system [4]. Therefore, the additional effort to remove many false positive links is accepted.

Nowadays, many software companies use issue tracking systems (ITS) to specify their requirements [15]. For open source projects, the usage of an ITS is a crucial point and de facto standard [18]. In ITS, the requirements text is unstructured and requirement issues are mixed with other issues for e.g. bug tracking, task and test management [18]. Furthermore, for many development activities, it is helpful to consider the links between requirements and source code *during development*, e.g. in maintenance tasks, for program comprehension and re-engineering [7,16].

If these links are created continuously during the development, e.g. after each commit performed by a developer, they can be used continuously. In these cases, the big effort of handling false positives, and thus bad precision is not practicable. Therefore, a trace link creation approach for links between unstructured requirements and code with perfect precision and good recall is needed.

Our interaction-based trace link creation approach (*IL*) aims at continuous link creation and usage. *IL* relies on developers manually selecting the requirement in their IDE before they start to work on it. Then, interactions recorded in the IDE are assigned to this requirement and the code files touched during interactions assigned to this requirement are used to create trace links. The approach is implemented in a corresponding tool[1].

In an initial study [10] based on open source data, we could show that *IL* links can have perfect precision and good recall (i.e. at least above 80%), if the developers use the requirements selection systematically. Since the initial recall of *IL* for one of the data sets used in this study was below 80%, we also used *source code structure* to improve the recall. Source code structure denotes relations between source code files such as references. Basically, we added further links by following the references of already linked files.

In a second study [11] with students as developers, the developers did not perform the manual selection of the requirements systematically. This lead to the creation of wrong trace links by our *IL* approach (precision of 43.0%, recall of 73.7% and $f_{0.5}$-measure of 0.469). To countervail the creation of wrong links, we came up with *wrong link detection techniques*. On the one hand, we used techniques based on data of the recorded interaction (e.g. interaction duration and frequency). On the other hand, we used techniques based on the source code structure. With these wrong link detection techniques, we could improve the precision of *IL* to 68.2% ($f_{0.5}$-measure of 0.624) [11]. However, the precision improvement also resulted in a decline of the recall to 46.5%. This is still not satisfying. Thus, we looked for a way to remove the error-prone manual selection of the requirement by the developers.

[1] https://se.ifi.uni-heidelberg.de/il.html, tool and data download.

The usage of issue identifiers (IDs) to link commits to requirements and bug reports is a common convention in open source projects [2,18,23]. Developers often create trace links themselves by providing issue IDs in commit messages [22]. Trace links can be created by linking all files affected by a commit with the requirement specified by the issue ID in the commit message. This is little effort and does not require the awareness for interaction recording. However, as confirmed by recent research, typically only 60% of the commits are linked [23]. Therefore, our idea is to combine the ID-based linking with interaction recording. Instead of using manually selected requirements, the issue IDs from developers' commit messages are used. All code files touched in the interactions before the commit are associated to the requirement identified through the issue ID.

As the students of our second study also used requirement issue IDs in their commit messages, we first simulated the combination of issue ID and *IL* retrospectively with the data of our previous study (without using the wrong link detection techniques). This directly improved the precision from 43.0% to 56.6% without affecting the recall.

This encouraged us to improve our approach with consideration of IDs in the commits. This new approach is called IL_{Com}. We applied IL_{Com} in a new study with students. Without further wrong link detection techniques, IL_{Com} had a precision of 84.9% and a recall of 67.3% ($f_{0.5}$-measure of 0.807). We also applied our wrong link detection techniques [11] and recall improvements [10] and could finally achieve a precision of 90.0% and a recall of 79.0% ($f_{0.5}$-measure of 0.876). Only using the issue IDs and the list of changed files from commits for link creation similar as described by [23] together with our wrong link detection and recall improvement techniques resulted in a precision of 67.5% and recall of 44.3% ($f_{0.5}$-measure of 0.611). For IR-created links using latent semantic indexing (LSI) [6], also including our wrong link detection, precision was 36.9% and recall 55.7% ($f_{0.5}$-measure of 0.396). Thus, in our new study we show that IL_{Com} achieves very good precision and recall and that it is much better in both precision and recall than the standard techniques.

The remainder of the paper is structured as follows. Section 2 introduces basics for evaluation of automatically created trace links, the projects used for the evaluation, our basic *IL* approach and former study results. Section 3 illustrates the details of the IL_{Com} approach and its implementation. It also reports on our retrospective preliminary study. Section 4 outlines the experimental design of our new study. Section 5 presents and discusses the results of our new study. Section 6 discusses the threats to validity and Sect. 7 related work. Finally, Sect. 8 concludes the paper and gives an outlook on our further research.

2 Background

In this section we introduce basics of trace link evaluation, describe the used study projects, introduce our *IL* approach and report about *IR* based trace link creation and results of our former studies.

2.1 Trace Link Evaluation

In the following, we sketch the basics of trace link evaluation as already described in our paper [11]. To evaluate approaches for trace link creation [3,5], a gold standard which consists of the set of all correct trace links for a given set of artifacts is important. To create such a gold standard, it is necessary to manually check whether trace links exist for each pair of artifacts. Based on this gold standard, precision and recall can be computed.

Precision (P) is the amount of correct links (true positives, TP) within all links found by an approach. The latter is the sum of TP and incorrect links (false positive, FP). Recall (R) is the amount of TP links found by an approach within all existing correct links. The latter is the sum of TP and false negative (FN) links:

$$P = \frac{TP}{TP + FP} \qquad R = \frac{TP}{TP + FN} \qquad F_\beta = (1 + \beta^2) \cdot \frac{P \cdot R}{(\beta^2 \cdot P) + R}$$

F_β-scores combine the results for P and R in a single measurement to judge the accuracy of a trace link creation approach. As shown in the equation, for F_β above, β can be used to weight P in favor of R and vice versa. In contrast to other studies, our focus is to emphasize P, but still consider R. Therefore, we choose $F_{0.5}$ which weights P twice as much as R. In addition, we also calculate F_1-scores to compare our results with others. For approaches using structured [8] and unstructured [19] data for trace link creation good R values are between 70 and 79% and good P values are between 30 and 49%.

2.2 Evaluation Projects

In the following, we describe the three projects we used for the evaluation of our approach. The first project is Mylyn a open source project which we used in our first *IL* evaluation [10]. In the project a plug-in to manage development tasks directly within the IDE is developed. We used the public accessible project's requirements and interaction log data stored in an ITS and source code in the Git version control system to create our data sets. We created two data sets using excerpts of the Mylyn project data from the years 2007 and 2012. They are called M_{2007} and M_{2012} in the following. Details can be found in [10].

The other two projects are university student projects we created one data set for each project using interaction recording of our tools, requirements managed in an ITS and source code in a version control system. Since the first university student project finished in 2017 and the second in 2018 the data sets for these project are called S_{2017} and S_{2018} respectively. Both are Scrum-oriented working with a real world customer in sprints. The first of them lasted from October 2016 to March 2017. We used it in the evaluation of a first *IL* improvement [11]. The second project lasted from October 2017 to March 2018. We used this project's data set S_{2018} in the actual evaluation. The details of the second data set are explained in Sect. 4.2.

The aim of the S_{2017} *project* was to develop a system to store and manage all health care reports for a patient in a single data base. The customer was the IT department of the university hospital. The aim of the S_{2018} project was to develop an Android-based indoor navigation app for students in university buildings. Typical use cases for such an app are navigating to the room of a certain lecture or finding any other point of interest efficiently. The customer was a mobile development company. In both projects, an adviser from our chair was involved. Seven students participated in the S_{2017} project and six students in the S_{2018} project. The projects were split in a corresponding number of sprints. In each of these sprints, one of the students acted as Scrum master and thus was responsible for all organizational concerns such as planning the development during the sprint and communicating with the customer.

For all requirement management-related activities, in both projects a Scrum Jira[2] Project was used. This included the specification of requirements in the form of user stories and the bundling of the stories in epics. An example of a user story in the navigation app project is *Show point to point route* and the corresponding epic of this story is *routing*. To assign the implementation of user stories to developers, sub-task issues were used. A sub-task comprises partial work to implement a user story, e.g. *Show route info box*. For the implementation, the developers used Git as version control system and the Webstorm[3] version of intelliJ in the first and the Eclipse IDE with the Android software development kit (SDK) in the second project. For both IDEs we provided plug-ins implementing our interaction recording tools. For the usage of Git in the S_{2018} project, there was an explicit guideline to use a Jira Issue ID in any commit message to indicate the associated Jira Issue. Although not directly required, the developers used this convention in the S_{2017} project as well.

In the first S_{2017} project, the developers used JavaScript as programming language which was requested by the customer. Furthermore, the MongoDB[4] NOSQL database and the React[5] UI framework were used. In the S_{2018} project, the customer provided a proprietary Java SDK of their own for the general use case to develop Android mobile navigation apps. The developers needed two sprints to understand the complexity of the SDK and to set up everything in a way to work efficiently on the implementation of requirements. The programming language for the logic and data management part was Java and the UI was implemented in Android's own XML based language.

In both projects at the beginning of the first sprint, we supported the developers with the installation and initial configuration of our interaction log recording tools. We also gave a short introduction on the implemented interaction-recording mechanism and how to use the tools during the project.

In Sect. 2.4 we summarize the previous evaluations and compare them with information retrieval based link creation. In Sect. 3.3 we use the S_{2017} project to

[2] https://www.atlassian.com/software/jira.
[3] https://www.jetbrains.com/webstorm/.
[4] https://www.mongodb.com/.
[5] https://reactjs.org/.

test our assumption that using issue IDs in commit messages improves the precision. In Sect. 4 we use the S_{2018} project to evaluate our new approach IL_{Com}.

2.3 IL Approach Overview

Figure 1 shows the overview of our *IL* approach consisting of three steps. In the first step *Interaction Capturing*, interaction events in the IDE of a developer are captured and associated to a requirement. We implemented our approach as plug-in for the IntelliJ IDE. For this, we extended an existing activity tracker plug-in to also track the interactions with requirements. In addition, we used the *Task & Context* functionality of IntelliJ to associate interactions with requirements. The developers could connect IntelliJ to the Jira project and the developers had to select the specific Jira issue with the UI of the *Task & Context* functionality when working on a requirement. As a result, the interaction log contained activation and deactivation events for requirement issues. These activation and deactivation events were used to allocate all interactions between the activation and deactivation event for a specific requirement to this specific requirement. Since the developers also used sub-task issues and sub-tasks describe details for implementing the requirement, we combined the interactions recorded for requirements and for the corresponding sub-tasks.

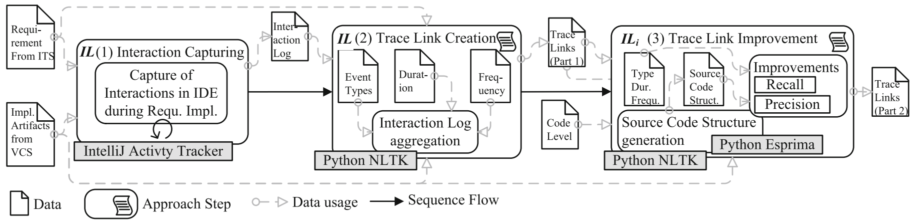

Fig. 1. *IL* Trace link creation overview: interaction capturing, trace link creation and improvement IL_i

In the second step *Trace Link Creation*, all interaction events captured for a requirement are used to generate trace links between the requirement and the source code files affected by the interactions. We did not consider files such as build configurations, project descriptions, readme files, meta-data descriptions, binaries etc. and files from 3rd parties such as libraries, as we focused on the code created by the developers. Interaction event-specific metadata like the event type (edit or select), the duration as the sum of all events' durations based on the interactions' time stamps for specific files and the frequency of how often an interaction occurred for a specific file were captured. The result of this second step is a list of trace links including the metadata aggregated from these interactions which is used as input for the third step *Trace Link Improvement*.

In this third step, precision is improved by removing potential wrong links using the interaction-specific metadata frequency, duration, and event type from the previous step. For frequency, duration, and event type, different settings are possible. Precision is also improved by using the source code structure, i.e. the references from one source code file to other source code files. In our P_{2017} study, we found that linking only source code files which are connected by source code structure with each other improves the precision significantly (*source code structure in story*) [11]. Finally, we also use the source code structure to improve the recall of *IL*. In this case, the source code structure of source code files which are already linked to a requirement is utilized. We add links by following the relations of the source code structure to other source code files up to a certain level [10].

In the following, we denote our *IL* approach as *IL* when applying the first two steps only and as IL_i when also applying the improvement techniques of the third step.

2.4 IR Based Link Creation and Previous Studies

To compare the results of our *IL* approach we also created links with information retrieval (*IR*). *IR* based link creation uses the textual content of documents and creates links based on textual similarity. Before document text content is processed by *IR* preprocessing of the textual content is performed. We performed all common *IR* preprocessing steps like stop word removal, punctuation, character removal, and stemming [1,3]. We also performed camel case identifier splitting (e.g. RouteInfoBox becomes Route Info Box), since camel case notation has been used in the source code [6]. In our studies we used the two most common *IR* techniques for trace link creation vector space model (VSM) and latent semantic indexing (LSI) [3,5,6]. The basic difference between these two *IR* techniques is that LSI can also consider synonyms of terms as similar whereas VSM only considers equal terms.

In the P_{2018} project the requirements were specified in German, but the source code files were in English. Thus we automatically translated the P_{2018} requirements using the googletrans Python library[6] before preprocessing and *IR* application. Since the user stories of both student projects P_{2017} and P_{2018} contained only short texts, the used threshold values for *IR* had to be set low. Source code structure-based precision and recall improvements (cf. Sect. 2.3) have also been applied to the *IR* (IR_i) and *IL* created trace links (IL_i).

Table 1 shows the results for *IR* and *IL* for our previous studies using the data sets explained in Sect. 2.2. When comparing the precision, recall and $f_{0.5}$-measures of IR_i and IL_i, IL_i clearly outperforms IR_i in all three data sets.

3 Commit Based Link Creation and $\mathbf{IL_{Com}}$

In this section, we introduce our commit-based variant of *IL*, called IL_{Com}. We provide an overview of all trace link creation techniques used in our new study.

[6] https://pypi.org/project/googletrans/.

Table 1. Results for *IR* and *IL* in previous studies

Approach[a]		Data Set	Pre-cision	Re-call	$F_{0.5}$	$F_{1.0}$	#Links[b]					#Sto-ries	Src Files	
							CE	TP	FP	GS	FN		Used	GS
IL_i	Interaction link creation with improvement	M_{2007}	1.000	0.929	0.985	0.963	2565	2565	0	2761	196	50	627	627
		M_{2012}	1.000	0.800	0.952	0.889	1126	1126	0	1408	282	50	363	702
		P_{2017}	0.682	0.465	0.624	0.553	148	101	47	217	116	13	63	91
IR	Information retrieval link creation	M_{2007}	0.310	0.248	0.295	0.275	1058	328	730	1324	996	41	200	585
		M_{2012}	0.298	0.558	0.328	0.388	920	274	646	491	217	35	169	444
		P_{2017}	0.343	0.161	0.280	0.219	102	35	67	217	182	9	17	91
IR_i	Information retrieval link creation with improvement	M_{2007}	0.386	0.440	0.396	0.411	3143	1214	1929	2761	1547	41	308	627
		M_{2012}	0.283	0.557	0.314	0.376	2766	784	1982	1408	624	35	354	702
		P_{2017}	0.351	0.217	0.312	0.268	134	47	87	217	170	9	21	91

[a] *IR* settings for the data sets are denoted as <IR-*model(similarity threshold)*>: M_{2007} $LSI(0.3)$, M_{2012} $LSI(0.5)$, P_{2017} $LSI(0.1)$

[b] created (CE), true positive (TP) $\widehat{=}$ correct, false positive (FP) $\widehat{=}$ wrong, gold standard (GS), false negative (FN) $\widehat{=}$ not found

This also includes the creation of trace links by only using commit data [22]. We also present the results of a preliminary retrospective simulated application of IL_{Com} to the data set P_{2017}.

3.1 $\mathbf{IL_{Com}}$

The difference between *IL* and IL_{Com} lies in the first interaction capturing step. IL_{Com} uses both recorded interactions and issue IDs in commit messages for link creation. In IL_{Com}, interactions are recorded until a developer performs a commit. If the commit message contains an issue ID, all recorded interactions are associated to this issue ID and the history of recorded interactions is cleared. If multiple issue IDs are contained in the commit message, the recorded interactions are associated to all issue IDs. If no issue ID is contained in the commit message, interaction recording continues until there is a commit with a commit message containing an issue ID. Clearly, this can impact precision and recall, as the commits without ID might be associated with another issue [9,13]. This will be discussed in Sect. 5.4. After the association of issue IDs with interactions has been obtained, link creation can be performed as described for *IL* in Sect. 2.3.

We implemented the interaction capturing for IL_{Com} for the P_{2018} project as plug-in for the Eclipse IDE. Our tool bundles all recorded interactions and uploads them to the Jira issue specified by the Jira issue ID in the commit message. The interaction events recorded by our tool comprise a time stamp, the type of interaction (select or edit), the part of the IDE in which the interaction occurred (e.g. editor, navigator, etc.), the file involved in the interaction, and a degree of interest (DOI) metric for the file. The DOI is a numerical value calculated for a file considering the number of interactions (frequency) and the type of interactions with the file, i.e. edit interactions are rated higher than select interactions [12].

3.2 Trace Link Creation Techniques

In the following we summarize the notations for the different link creation (*IR*, *IL*, *ComL* and IL_{Com}) and improvement techniques (shown by subscript i):

IR denotes the approach for link creation by information retrieval and $\boldsymbol{IR_i}$ denotes that also source code structure based improvement techniques have been applied (cf. Sect. 2.4).

IL denotes the approach for link creation by using the recorded interactions and $\mathbf{IL_i}$ denotes that also interaction-specific metadata and source code structure based improvement techniques have been applied (cf. Sect. 2.3).

ComL denotes the approach for link creation by using the issue IDs from commit messages and the files contained in the commits and $\boldsymbol{ComL_i}$ denotes that also source code structure based improvement techniques have been applied.

$\boldsymbol{IL_{Com}}$ denotes the approach for link creation by using the recorded interactions and the issue IDs from commit messages and $\boldsymbol{IL_{Com\_i}}$ denotes that also interaction-specific metadata and source code structure based improvement techniques have been applied (cf. Sect. 3.1).

3.3 Retrospective Study

As described in the introduction, we analyzed the P_{2017} project data set regarding IDs in commit messages. We found that there were significantly more commits with issue IDs (per developer) than there were activation and deactivation events in the recorded interaction logs. For one developer, the processing of 18 requirements was recorded in the interaction logs, but there were 71 commits with requirement issue IDs for the same developer in Git. This does not directly indicate that the interaction log recording is wrong, since it is possible that a developer performed multiple commits for one requirement successively. However, after a random check of the time span of interaction recording for two requirements we found that there were commits with different issue IDs in this time span. This encouraged us to analyze the data further and thus simulate retrospectively the application of IL_{Com}.

Table 2. 2017 project results: precision and recall for created trace links

Approach		Precision	Recall	$F_{0.5}$	$F_{1.0}$	#Links					#Stories	#Subtasks	Src Files	
						CE	TP	FP	GS	FN			Used	GS
IL	Interaction link creation	0.430	0.737	0.469	0.543	372	160	212	217	57	19	98	89	91
IL_i	With improvement	**0.669**	0.465	**0.615**	0.549	151	101	50	217	116	13	72	63	91
ComL	Commit link creation	0.620	0.465	0.581	0.532	163	101	62	217	116	19	98	78	91
$ComL_i$	With improvement	**0.659**	0.401	**0.584**	0.499	132	87	45	217	130	11	66	59	91
IL_{Com}	Inter. and commit link creation	0.566	0.733	0.593	0.639	281	159	122	217	58	19	98	86	91
IL_{Com_i}	With improvement	**0.736**	0.539	**0.686**	0.622	159	117	42	217	100	13	72	63	91

Table 2 shows the results for our retrospective study with the data from the data set P_{2017}. We created the trace links by the different approaches as described in the following. For *ComL*, we created links for all commits with requirement issue IDs in the commit message from the requirement referenced by the ID to all source code files of the commit. For IL_{Com}, we used the interactions recorded for *IL* and the commits with issue IDs. We ordered the Git commits with requirement issue IDs and the interaction log recording by time. All interaction log recordings between two commits with issue IDs are assigned to the issue from the second commit. Since there were also commits without issue ID which we just ignored in our evaluation, this kind of interaction log recordings to commit assignment is not perfect. If a developer just did not add an issue ID in a commit, interactions are assigned wrongly and precision is impaired. This simulates retrospectively the application of IL_{Com}.

Table 2 always shows the best achieved $f_{0.5}$-measure within all performed settings for an approach. Moreover, the overall best values for precision and $f_{0.5}$-measure are highlighted. IL_{Com_i} has a precision of 73.6%, a recall of 53.9% and a $f_{0.5}$-measure of 0.686 which outperforms the precision and recall of all other approaches. This confirmed our idea that *IL* can be combined with the use of issue IDs from commit messages.

4 Experiment Design

In this section, we describe the details of our new study starting with the research questions and the description of how we created the trace links and compared the results with our former studies in Sect. 4.1, followed by the description of the data sources in Sect. 4.2 and and the gold standard creation in Sect. 4.3.

4.1 Research Questions

The research questions we answer in our study are:

RQ$_1$: *What is the precision and recall of IL_{Com}- and IL_{Com_i}-created trace links?* Our hypothesis was that the initial precision of IL_{Com} improves, compared to our P_{2017} study, since there is no additional effort for requirement selection by developers. For IL_{Com_i} compared to IL_{Com}, we expected a further precision improvement.

RQ$_2$: *What is the precision and recall of ComL- and $ComL_i$-created trace links?* Our hypothesis was that precision and recall are worse than the precision of IL_{Com}- and IL_{Com_i}-created links respectively, as the latter uses more information (the interactions).

RQ$_3$: *What is the precision and recall of IR- and IR_i-created trace links?* Our hypothesis was that *IR* has a significantly worse precision and similar recall in comparison to IL_{Com}.

The overall goal of this new study is to evaluate, whether the interaction and commit based link creation by IL_{Com} improves the precision compared to the only interaction based link creation by IL (RQ_1). Moreover, we also would like to investigate whether recording and using interactions outperforms link creation, which relies on commit data only (RQ_2). Finally, we also compare the results of IL_{Com}-created links with IR, since IR serves as a baseline for automated link creation and for the comparison with our previous studies (RQ_3).

4.2 Data Sources

In our evaluation we used three different data sources which are described in the following.

Source Code in the Git Version Control System. The Git repository comprises 406 commits. 226 commits (55.67% of all commits) did contain a Jira issue ID which is a similar proportion as reported by others [22]. We excluded the same file types from the Git repository as for IL (cf. Sect. 2.3).

We used the first 395 commits in the Git Repository for link creation. The 395th commit is the commit for the finish of the project's last sprint. Commits after the 395th commit did not contain issue IDs and were performed to refactor the source code to the customer's needs after the final project presentation. The Git repository for the 395th commit contained 40 java and 26 xml files.

Requirements as Issues in Jira. After the project was finished, there were 23 story issues in the Jira project. However, three of the story issues did not specify requirements, but testing and project organization. Therefore, we removed these three stories from our evaluation. Furthermore, the processing status of 3 story issues was unresolved at the end of the project and in addition all sub-tasks of these 3 unresolved stories where unresolved as well. Therefore, we also removed these 3 stories and their interaction recordings from our evaluation and used only the 17 remaining stories and their 74 sub-tasks along with their interaction recordings.

Interaction Recordings. The interaction recordings for the 17 stories and 74 sub-tasks comprise 6471 interaction events separated in 205 commits. After removing interaction events whose files were out of scope as described previously (cf. Sect. 2.3), 4012 interaction events were left in the interaction recordings and used for link creation.

4.3 Gold Standard Creation

The gold standard creation was performed in March 2018 by the 6 developers of the project between the finish of the last sprint and the final presentation to the customer. The developers vetted link candidates between requirements and the source code files in the actual version (395th commit) in the projects Git repository.

The developers vetted the links based on their involvement in the sub-tasks of a requirement. If there were two developers with an equal amount of sub-tasks, both vetted the links and only the links vetted as correct by both were used in the gold standard. For each developer, a developer-specific interactive questionnaire spreadsheet with all link candidates to vet was generated. This contained for each requirement, all possible link candidates to all 66 source code files. The vetting resulted in 309 gold standard trace links, where each requirement and each code file was linked at least once.

5 Results

This section reports the results of our evaluations and answers the RQs.

Table 3. Results for IL_{Com} and IL_{Com_i} with different settings

Approach		Setting[a]	Precision	Recall	$F_{0.5}$	$F_{1.0}$	#Links					Src Files	
							CE	TP	FP	GS	FN	Used	GS
IL_{Com}	Default interaction link creation	none	0.849	0.673	0.807	0.751	245	208	37	309	101	58	66
IL_{Com_i}	Interaction type improvement	*T:e*	0.904	0.460	0.758	0.609	157	142	15	309	167	58	66
IL_{Com_i}	Interaction type improvement	*T:s*	0.829	0.282	0.597	0.420	105	87	18	309	222	37	66
IL_{Com_i}	Duration improvement	*D10*	0.885	0.521	0.776	0.656	182	161	21	309	148	52	66
IL_{Com_i}	Duration improvement	*D60*	0.901	0.411	0.727	0.564	141	127	14	309	182	50	66
IL_{Com_i}	Frequency improvement	*F2*	0.813	0.463	0.706	0.590	176	143	33	309	166	54	66
IL_{Com_i}	Frequency improvement	*F10*	0.850	0.311	0.631	0.455	113	96	17	309	213	40	66
IL_{Com_i}	Source code structure in story imp.	*Sis*	0.904	0.485	0.771	0.632	166	150	16	309	159	40	66
IL_{Com_i}	Selected improvement tech. setting	*T:e,s; Sis;CS*	**0.900**	0.790	**0.876**	0.841	271	244	27	309	65	62	66

[a] $T{:}e|s = Type{:}edit|select$, $D10|D60 = dur. >= 10|60$ $sec.$, $F2|10 = freq. >= 2|10$, Sis = *Source code structure in story*, CS = *Source code structure*

5.1 Answer to RQ1: Comparison of IL and $\mathbf{IL_{Com}}$

Table 3 shows the results for IL_{Com} and for different settings for IL_{Com_i}. IL_{Com} has a precision of 84.9% and a recall of 67.3% and thus a $f_{0.5}$-measure of 0.807. Similar to our P_{2017} study, we evaluated different settings for our improvement techniques (cf. first column of Table 3) [11]. Initially, we investigated the different wrong link detection techniques in isolation and then combined different techniques to achieve the overall best precision improvement. On this best precision result, we also applied our source code structure-based recall improvement. The last row of Table 3 shows this best case of IL_{Com_i}. For this, the setting was to use the type select and edit (T:e,s), to restrict the source code files to be connected with each other by code structure in the story (Sis) and to use the code structure to improve recall (CS). In this best case, IL_{Com_i} has a precision of 90.0% and a recall of 79.0% and thus a $f_{0.5}$-measure of 0.876. Thus, IL_{Com_i} improves precision by 5.1%, recall by 22.7% and $f_{0.5}$-measure by 0.069 compared to IL_{Com}.

5.2 Answer to RQ2: Comparison of $\mathbf{IL_{Com}}$ and ComL

Table 4 shows the results for *ComL* and $ComL_i$ and for comparison also the previously reported results of IL_{Com}. *ComL* has a precision of 66.8% and a recall of 41.7% and thus a $f_{0.5}$-measure of 0.597. For $ComL_i$, we first applied the *source code structure in story* precision improvement followed by *source code structure* recall improvement. $ComL_i$ has a precision of 67.5% and a recall of 44.3% and thus a $f_{0.5}$-measure of 0.611. In comparison to IL_{Com} and IL_{Com_i}, precision, recall, and $f_{0.5}$-measure are worse respectively.

Table 4. Results for *ComL*, $ComL_i$ and comparison with IL_{Com}

Approach[a]		Pre-cision	Re-call	$F_{0.5}$	$F_{1.0}$	#Links					Src Files	
						CE	TP	FP	GS	FN	Used	GS
IL_{Com}	Inter. and commit link creation	0.849	0.673	0.807	0.751	245	208	37	309	101	58	66
IL_{Com_i}	With improvement	0.900	0.790	0.876	0.841	271	244	27	309	65	62	66
ComL	Commit link creation	0.668	0.417	0.597	0.514	193	129	64	309	180	59	66
$ComL_i$	With improvement	0.675	0.443	0.611	0.535	203	137	66	309	172	61	66

[a] For the application of improvement techniques the best case is shown

5.3 Answer to RQ3: Comparison of $\mathbf{IL_{Com}}$ and IR

Table 5 shows the results for *IR* and IR_i and for comparison also the previously reported results of IL_{Com} and IL_{Com_i}. *IR* has a precision of 33.5% and a recall of 49.2% and thus a $f_{0.5}$-measure of 0.358. For P_{2018}, IR_i has a precision of 36.9% and a recall of 55.7% and thus a $f_{0.5}$-measure of 0.396. IR_i improves precision by 3.4%, recall by 6.5% and $f_{0.5}$-measure by 0.038 compared to *IR*. In comparison to IL_i and IL_{Com_i}, precision, recall, and $f_{0.5}$-measure is worse respectively. For all data sets, IL_i outperforms IR_i. The *IR* results for our former projects are quite similar and similar to other studies as well [19].

Table 5. Results for *IR*, IR_i and comparison with IL_{Com} and IL_{Com_i}

Approach[a]		Pre-cision	Re-call	$F_{0.5}$	$F_{1.0}$	#Links					#Sto-ries	Src Files	
						CE	TP	FP	GS	FN		Used	GS
IL_{Com}	Inter. and commit link creation	0.849	0.673	0.807	0.751	245	208	37	309	101	17	58	66
IL_{Com_i}	With improvement	**0.900**	0.790	**0.876**	0.841	271	244	27	309	65	17	62	66
IR	Information retrieval link creation	0.335	0.492	0.358	0.398	454	152	302	309	157	16	60	66
IR_i	With improvement	0.369	0.557	0.396	0.444	466	172	294	309	137	16	64	66

[a] *IR* settings are denoted as <IR-*model(similarity threshold)*> : *VSM(0.2)*

5.4 Discussion

Precision and recall of IL_{Com} are better than *IL*. When looking at all studies we performed, it can be seen that *IL* and IL_{Com} outperform all other link creation approaches, i.e. *IR*- and commit-based link creation *ComL* (cf. Table 1 in Sect. 2.4). The fact that *IR* link creation between unstructured requirements in ITS and source code is worse than in structured requirement cases is reported by others [3,8,19]. This is also confirmed by our three studies (cf. results for *IR* in Tables 1 and 5) and was one of our initial motivations for the development of *IL*.

There are several possible reasons for the worse behaviour of *ComL* in comparison to IL_{Com}. It is interesting that the precision of *ComL* is roughly 60% in the retrospective study and in the new study. That means the issue IDs given by the developers are only partly correct. This observation is similar to research within developers' commits behavior and the contents of commits [9,13]. These studies report about tangled changes, that is a commit often comprises multiple unrelated issues. Also, we observed that developers manually excluded files in one commit, which were correct in the gold standard and then included these files in a follow-up commit. A reason for this behavior could be a change of the requirement during the project time. Thus, the exclusion behavior was correct when the commit was performed, but was wrong for the final state of the requirement. The reasons for the worse recall of *ComL* in comparison to IL_{Com} could be select interactions. Select interactions are not detected by commits. These missed files also affect the application of source code structure-based recall improvement.

The improvement techniques developed in our last studies also proved to be reasonable in this new study. Moreover, the improvement techniques also performed well for links created with *IR* and *ComL*. By applying our wrong link detection techniques, the precision is improved, independent of how the links were created. As wrong links detection techniques impair recall, we apply source code-structured based recall improvement. The improvement of recall by using the source code structure worked reasonable for *IL* in the last two studies and is outperformed in this new study. The application of recall improvement in this new study resulted in the best overall recall for the complete studies.

Altogether we showed that the creation of links with interaction and commit data by IL_{Com_i} achieves very good precision and recall. This confirms our assumption that the additional effort of manually selecting the requirement to work on caused the bad precision of *IL* in our previous P_{2017} study. We think that precision and recall can be even better, if developers directly use the created links during the projects, as in the Mylyn project. The use will likely motivate developers to use interaction logging and commit IDs carefully.

6 Threats to Validity

As described in our previous study [11] the internal validity is threatened as manual validation of trace links in the gold standard was performed by the students working as developers in a project context of our research group. However, this ensured that the experts created the gold standard. Also the evaluation of the links was performed after the project had already been finished so that there was no conflict of interest for the students to influence their grading.

When comparing the results achieved with our approach to *IR*, the setup of the *IR* algorithms is a crucial factor. Regarding preprocessing, we performed all common steps including the identifier splitting which is specific to our used data set. However, the low threshold values impair the results for the precision of *IR*. Therefore, further comparison of *IL* and *IR* in which higher threshold values are possible (e.g. with more structured issue descriptions) is necessary.

The external validity depends on the availability of interaction logs and respective tooling and usage of the tooling by developers. The generalizability based on one student project is clearly limited. Although explicitly requested, not all commits contained a Jira issue ID in the commit messages. This affects the resulting association of recorded interaction logs to requirement issues and thus the created trace links. However, the percentage of commits with issue IDs is similar as reported for other projects [22]. This indicates that the results of our evaluation might also apply for industry projects.

7 Related Work

In our previous papers [10,11], we already discussed related work on *IR*, interaction logging and the assessment of interaction recording quality which is shortly summarized in the following: The systematic literature review of Borg on *IR* trace link creation [3] gives an overview of *IR* usage and results. In [14], Konopka uses interaction logs to detect relations between code files and in [24], Soh showed with an observation study that observed interaction durations do not always correspond to recorded interaction durations.

In [20], Omoronyia published an approach in which interactions are used to visualize and navigate trace links. In a follow up paper [21] of the same authors, they also use interactions for trace link creation. They consider developer collaboration and rank interaction events. Their approach achieves a precision of 77% in the best case which is still not as good as our results for IL_{Com}.

In [22], Rath report about a data set *Ilm7* they created from seven open source projects for the purpose of evaluating traceability research. They used the issue IDs in commit messages to link issues to code files. They report that only 60% of the commits contain an issue ID.

In their follow-up work [23], they use the *Ilm7* data set to train different machine learning classifiers to countervail the problem of commits without issue IDs. To train their classifiers, they not only used the files and issue IDs from commits, but also textual similarity (*IR*) between different artifacts (i.e. the commit message text, the issue text, the source code text) and further data like developer-specific information. In their final experiment, they used the trained machine learning classifiers to identify the matching issues for commits without issues and achieved an averaged recall of 91.6% and precision of 17.3%. A direct comparison with *IR*-based link creation is missing. However, since these results are quite similar to what others have achieved with relying on *IR* [19] and ITS data only, it seems that the usage of *IR* to train machine learning classifiers results in the same low precision values as when relying on *IR* only. When directly comparing their results with the results achieved by IL_{Com} in this study (recall of 79.0% and precision of 90%), it is clear that for our research goal of precision optimization IL_{Com} is far superior.

8 Conclusion and Outlook

In this paper, we investigated the precision and recall of our interaction-based trace link creation approach IL_{Com}. In contrast to our previous studies, we changed the implementation of our interaction log recording tool. With the new implementation, we reduce the additional effort for developers to assign interaction log recordings to requirements and removed the need for interaction log recording awareness.

Our new approach and tool build on the common practice to specify issue IDs in commit messages. It uses these issue IDs from commit messages to assign interaction log recording to requirements. IL_{Com} has a precision of 90.0% and recall of 79.0% which outperforms the results of our previous P_{2017} study (precision of 68.2% and recall of 46.5%). Thus, precision is not perfect, but we think that this is a very good basis for continuous link creation and usage. Furthermore, the new approach is applicable also where developers are not particularly interested in interaction recording. We showed that our new approach outperforms *IR* and purely commit-based linking and is superior to current machine learning based approaches as well [23]. Clearly, it is interesting to confirm this with further studies and to study whether this also holds for more structured requirements where *IR* is typically used. Another important step for applicability in practice is to investigate the maintenance of links such as [17].

Acknowledgment. We thank the students of the projects for their effort.

References

1. Baeza-Yates, R., de Ribeiro, B.A.N.: Modern Information Retrieval, 2nd edn. Pearson Addison-Wesley, Boston (2011)
2. Bird, C., Rigby, P.C., Barr, E.T., Hamilton, D.J., Germán, D.M., Devanbu, P.T.: The promises and perils of mining git. In: MSR (2009)
3. Borg, M., Runeson, P., Ardö, A.: Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. ESE **19**(6), 1565–1616 (2013)
4. Briand, L., Falessi, D., Nejati, S., Sabetzadeh, M., Yue, T.: Traceability and SysML design slices to support safety inspections. ToSEM **23**(1), 9 (2014)
5. Cleland-Huang, J., Gotel, O.C.Z., Huffman Hayes, J., Mäder, P., Zisman, A.: Software traceability: trends and future directions. In: ICSE/FOSE. ACM (2014)
6. De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Recovering traceability links in software artifact management systems using information retrieval methods. ToSEM **16**(4), 13 (2007)

7. Ebner, G., Kaindl, H.: Tracing all around in reengineering. IEEE Softw. **19**(3), 70–77 (2002)
8. Hayes, J., Dekhtyar, A., Sundaram, S.: Advancing candidate link generation for requirements tracing: the study of methods. TSE **32**(1), 4–19 (2006)
9. Herzig, K., Zeller, A.: The impact of tangled code changes. In: MSR. IEEE (2013)
10. Hübner, P., Paech, B.: Using interaction data for continuous creation of trace links between source code and requirements in issue tracking systems. In: Grünbacher, P., Perini, A. (eds.) REFSQ 2017. LNCS, vol. 10153, pp. 291–307. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-54045-0_21
11. Hübner, P., Paech, B.: Evaluation of techniques to detect wrong interaction based trace links. In: Kamsties, E., Horkoff, J., Dalpiaz, F. (eds.) REFSQ 2018. LNCS, vol. 10753, pp. 75–91. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77243-1_5
12. Kersten, M., Murphy, G.C.: Using task context to improve programmer productivity. In: SIGSOFT/FSE. ACM (2006)
13. Kirinuki, H., Higo, Y., Hotta, K., Kusumoto, S.: Hey! Are you committing tangled changes? In: ICPC. ACM (2014)
14. Konopka, M., Navrat, P., Bielikova, M.: Poster: discovering code dependencies by harnessing developer's activity. In: ICSE. ACM (2015)
15. Maalej, W., Kurtanovic, Z., Felfernig, A.: What stakeholders need to know about requirements. In: EmpiRE. IEEE (2014)
16. Mäder, P., Egyed, A.: Do developers benefit from requirements traceability when evolving and maintaining a software system? Empir. SE **20**(2), 413–441 (2015)
17. Maro, S., Anjorin, A., Wohlrab, R., Steghöfer, J.: Traceability maintenance: factors and guidelines. In: ASE (2016)
18. Merten, T., Falisy, M., Hübner, P., Quirchmayr, T., Bürsner, S., Paech, B.: Software feature request detection in issue tracking systems. In: IEEE RE Conference (2016)
19. Merten, T., Krämer, D., Mager, B., Schell, P., Bürsner, S., Paech, B.: Do information retrieval algorithms for automated traceability perform effectively on issue tracking system data? In: Daneva, M., Pastor, O. (eds.) REFSQ 2016. LNCS, vol. 9619, pp. 45–62. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30282-9_4
20. Omoronyia, I., Sindre, G., Roper, M., Ferguson, J., Wood, M.: Use case to source code traceability: the developer navigation view point. In: IEEE RE Conference (2009)
21. Omoronyia, I., Sindre, G., Stalhane, T.: Exploring a Bayesian and linear approach to requirements traceability. IST **53**(8), 851–871 (2011)
22. Rath, M., Rempel, P., Mäder, P.: The IlmSeven dataset. In: RE Conference (2017)
23. Rath, M., Rendall, J., Guo, J.L.C., Cleland-Huang, J., Mäder, P.: Traceability in the wild: automatically augmenting incomplete trace links. In: ICSE (2018)
24. Soh, Z., Khomh, F., Guéhéneuc, Y.G., Antoniol, G.: Noise in Mylyn interaction traces and its impact on developers and recommendation systems. ESE **23**(2), 645–692 (2018)