Evaluation of Techniques to Detect Wrong Interaction Based 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 projects where trace links are created and used continuously during the development, it is important to support developers with an automatic trace link creation approach with high precision. In our previous study we showed that our interaction based trace link creation approach achieves 100% precision and 80% relative recall and thus performs better than traditional IR based approaches. [**Question/problem**] In this study we wanted to confirm our previous results with a data set including a gold standard created by developers. Moreover we planned further optimization and fine tuning of our trace link creation approach. [**Principal ideas/results**] We performed the study within a student project. It turned out that in this study our approach achieved only 50% precision. This means that developers also worked on code not relevant for the requirement while interactions were recorded. In order to improve precision we evaluated different techniques to identify relevant trace link candidates such as focus on edit interactions or thresholds for frequency and duration of trace link candidates. We also evaluated different techniques to identify irrelevant code such as the developer who created the code or code which is not related to other code in an interaction log. [**Contribution**] Our results show that only some of the techniques led to a considerably improvement of precision. We could improve precision almost up to 70 % while keeping recall above 45% which is much better than IR-based link creation. The evaluations show that the full benefits of an interaction based approach highly depend on the discipline of the developers when recording interactions for a specific requirement. Further research is necessary how to support the application of our approach in a less disciplined context.

Keywords: Traceability · Interaction · Requirement · Source code Precision

1 Introduction

Existing trace link creation approaches are typically based on information retrieval (IR) and on structured requirements like use cases or user stories. Also, they often

E. Kamsties et al. (Eds.): REFSQ 2018, LNCS 10753, pp. 75–91, 2018.
https://doi.org/10.1007/978-3-319-77243-1_5

focus on links between requirements [2]. It is known that precision of IR created links is often not satisfying [8] for their direct usage even in the case of structured requirements. Thus, handling of false positive IR created trace links requires extra effort in practice which is even a research subject on its own [7,9,19].

Still, the research focus in RE is to improve recall, since security critical domains like the aeronautics and automotive industry require complete link sets and thus accept the effort to remove many false positives [3]. These links are created periodically, when needed for certification to justify the safe operation of a system.

However, in many companies requirements are managed in issue tracking systems (ITS) [15]. For open source projects ITS are even the de facto standard for all requirements management activities [17]. In ITS the requirements text is unstructured, since ITS are used for many purposes, e.g. development task and bug tracking in addition to requirement specification. This impairs the results of IR-based trace link creation approaches [18]. Furthermore, for many development activities it is helpful to consider links between requirements and source code during development, e.g. in maintenance tasks and for program comparison [16]. If these links are created continuously, that means after each completion of an issue, they can be used continuously during the development. In these cases, large effort for handling false positives and thus, bad precision is not practicable. Therefore, a trace link creation approach for links between unstructured requirements and code is needed with perfect precision and good recall. Recall values are reported as good above 70% [9].

In a previous paper [10] we provided such a trace link creation approach (called IL in the following) based on interaction logs and code relations. Interaction logs capture the source code artifacts touched while a developer works on an issue. Interaction logs provide more fine-grained interaction data than VCS change logs [6]. Code relations such as references between classes provide additional information. In a previous study using data from an open source project we showed that our approach can achieve 100% precision and 80% relative recall and thus performs much better than traditional IR based approaches [11]. As there are no open source project data available with interaction logs and a gold standard for trace links, we only could evaluate recall relative to all correct links found by our approach and IR.

In contrast to the previous paper we now present a study based on interaction log data, requirements and source code from a student project. We used a student project in order to be able to create a gold standard with the help of the students. This enabled the calculation of the recall against the gold standard.

The presented study consists of two parts. In the first part we calculated precision and real recall values for our IL approach. The first results of the study showed that IL has only around 50% precision. We therefore evaluated the wrong links identified by IL. We found out that these links were caused by developers not triggering the interaction recording for requirements correctly. They worked on different requirements without changing the requirement in the IDE. Thus, all trace links were created for one requirement.

In consequence, in the second part of our study, we evaluated different techniques to improve precision by identifying relevant trace link candidates such as focus on edit interactions or thresholds for frequency and duration of interactions. We also evaluated different techniques to identify irrelevant code such as the developer who created the code, or code which does not refer to other code in an interaction log. In the best cases we could improve the precision up to almost 70% with still reasonable recall above 45%.

The remainder of this paper is structured as follows. Section 2 gives a short introduction into the evaluation of trace link creation approaches and the project used for the evaluation. Section 3 presents our interaction based trace link creation approach. Section 4 introduces the experimental design along with the creation of data sets for our study, states the research questions and introduces the improvement techniques to detect wrong trace links for our approach developed in this study. In Sect. 5 we present the results of the study and answer the research questions including a discussion. Section 6 discusses the threats to validity of the study. In Sect. 7 we discuss related work. Section 8 concludes the paper and discusses future work.

2 Background

In this section we introduce the basics of trace link evaluation and the study context.

2.1 Trace Link Evaluation

To evaluate approaches for trace link creation [2,8] 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 not correct links (false positive, FP). Recall (R) is the amount of TP links found by an approach within all existing correct links (from the gold standard). 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. In our previous paper [11] information about typical values of P and R in settings using structured [9] and unstructured [18]

data for trace link creation approaches can be found. Based on these sources for unstructured data good R values are between 70 and 79% and good P values are between 30 and 49%.

2.2 Evaluation Project

Due to the labor intensity of creating a trace link gold standard often student projects are used [5]. In the following we describe the student project in which we recorded the interactions, the application of the used tools and how we recorded the interactions. The project lasted from Oct. 2016 to March 2017 and was performed Scrum oriented. Thus it was separated into seven sprints with the goal to get a working product increment in each sprint. The projects aim was to develop a so called *master patient index* for an open ID oriented organization of health care patient data. A typical use case for the resulting product would be to store and manage all health care reports for a patient in a single data base. The project involved the IT department of the university hospital as real world customer. Further roles involved were the student developers and a member of our research group with the role of a product owner. Seven developers participated in the project. In each of the sprints one of developer acted as scrum master.

All requirements related activities were documented in a Scrum Project of the ITS JIRA[1]. This included the specification of requirements in the form of user stories and the functional grouping of the requirements as epics. For instance the epic *Patient Data Management* comprised user stories like *View Patients* or *Search Patient Data*. Complex user stories in turn comprised sub-tasks documenting more and often technical details. For instance the *Search Patient Data* user story comprised the sub-tasks *Provide Search Interface* or *Create Rest Endpoint*. The project started with an initial vision of the final product from the customer and was broken down by the developers using the scrum backlog functionality of JIRA to a set of initial user stories which evolved during the sprints.

For implementation the project used JavaScript which was requested by the customer. Furthermore the MongoDB[2] NOSQL database and the React[3] UI framework were used. The developers used the Webstorm[4] version of IntellJ IDE along with Git as version control system. Within the JIRA project and the JavaScript source code we also applied our feature management approach [21]. A feature in this project corresponded to an epic. This approach ensures that all artifacts are tagged with the name of the feature they belong to. So that a user story is tagged with the epic it corresponds to, but also the sub-tasks of the user stories and the code implementing the user story are tagged.

The developers installed and configured IntelliJ plug-ins we used for interaction recording (cf. Sect. 3) and were supported whenever needed. They got a short introduction about interaction recording and associating requirements

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

and source code files. The plug-ins recorded all interactions in the IDE in locally stored csv and xml files. The developers were asked to send us their interaction log files by email after each sprint on voluntary basis so that we had the possibility to check the plausibility of the recorded interactions. In the first sprints some of the developers had problems with activating interaction recording and using the desired IntelliJ plug-in to interact with requirements. After detecting such problems we explained it to them and asked them to solve these problems for the processing of the next sprint. However some of the developers only sent their interaction logs once or twice in the final project phase. Therefore four of the seven log files received were not usable for our evaluation. One was almost empty due to technical problems, in the other three only a very low number of requirements were logged. The corresponding developers stopped to record changes to requirements at a certain point in time and thus all following interactions were associated with the last activated requirement. We used the three correctly recorded interaction logs to apply our IL approach. Overall the interaction logs of the three developers contained more than two million log entries. The developers recorded these interactions while working on 42 distinct user stories and sub-tasks and touching 312 distinct source code files.

3 Interaction Based Trace Link Approach

Figure 1 shows our interaction based trace link creation approach (*IL*) and the improvement step IL_i. First we use an IDE Plug-in to capture the interactions of the developer while working on requirements and code. In a second step trace links are created between requirements and code based on the interactions. The last step is an improvement step that uses source code structure and interaction log data. In the following we explain the steps in more detail.

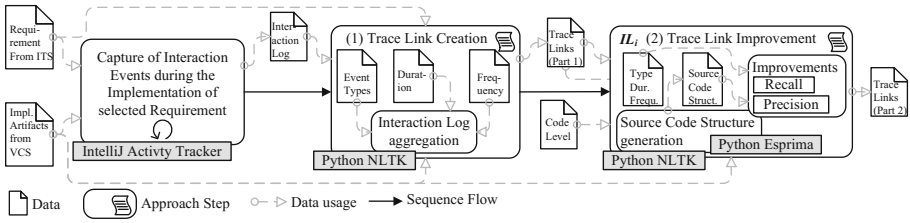

Fig. 1. *IL* trace link creation overview: interaction capturing, trace link creation and improvement IL_i

3.1 Interaction Logs

In contrast to our last study we used the IntelliJ IDE[5] and implemented the first interaction capturing step of our IL approach with two IntellJ Plug-ins:

1. To log interactions we used the *IntellJ Activity Tracker* Plug-in which we modified to our needs. We extended the Plug-ins ability to track the interactions with requirements. The only action to be performed by the developers for this plug-in was to activate it once. After this all interactions within the IDE of the developer were recorded, comprising a time stamp, the part of the IDE and the type of interactions performed. The most important part of the IDE for us are the editor for the source code, the navigator which displays a structural tree of all resources managed by the IDE, and dialogs which are often involved in high level actions like committing to Git and performing JIRA Issue related actions. The interaction types can be low level interactions like editor keystrokes, but also high level interactions (selected from the context menu) like performing a refactoring or when committing changes to Git.
2. To associate interactions with requirements the *Task & Context* IntellJ functionality was used. The developers connected this Plug-in with the JIRA project. When working on a requirement the developers selected the specific JIRA issue with the *Task & Context* functionality. When committing their code changes to the Git repository the *Task & Context* plug-in supported the finishing of the respective JIRA issue.

The following listing shows two abridged log entries as created by the modified version of the activity tracker tool.

```
1  2016-10-04T10:14:50.910;dev2;Action;EditorSplitLine;ise;Editor;
       /git/Controller.js;
2  2016-10-13T13:28:26.414;dev2;Task Activation;ISE2016-46:Enter Arrays;
       ise;
```

The first log entry is a typical edit interaction starting with a time stamp, the developers user name, the kind of performed action, the performed activity (which is entering a new line), the used Git project, the involved component of the IDE (*editor*) and the used source code file (*/git/Controller.js*). The second log entry shows an interaction with a user story from JIRA including its issue ID and name (*ISE2016-46:Enter Arrays*).

3.2 Trace Link Creation and Improvement

The actual IL trace link creation has been implemented in our Python NLTK[6] based tool. As shown in Fig. 1 in step *IL-Trace Link Creation (1)* interactions of the same requirements are aggregated and trace link candidates are created using the data of interaction logs, the source code touched by the interactions extracted from the version control system and the requirements from the ITS. The candidates relate the requirement associated to the interaction and the source code touched in the interaction.

[5] https://www.jetbrains.com/idea/.
[6] http://www.nltk.org/.

In the *IL$_i$-Trace Link Improvement (2)* step source code structure and interaction log data such as duration and frequency are used to improve recall (cf. Fig. 1). The source code structure based improvement of this step has been implemented with the Esprima[7] JavaScript source code parser. With source code structure we denote the call and data dependencies between code files and classes [14]. Using the code structure to improve trace link creation is part of traceability research [13]. In our previous study we added additional links to a requirement by utilizing the code structure of source code files already linked to the requirement [11]. As we aim at trace links with perfect precision this recall improvement only makes scene, if the trace links have excellent precision. Otherwise utilization of code structure might increase recall but very likely also decrease precision.

In this paper we also use the code structure to support precision by utilizing the relations between source code files involved in the interaction logs of one requirement (cf. Sect. 4.4).

4 Experiment Design

In this section we describe the details of our study (cf. Fig. 2), in particular wrt. the data sets and the techniques to detect wrong interaction links.

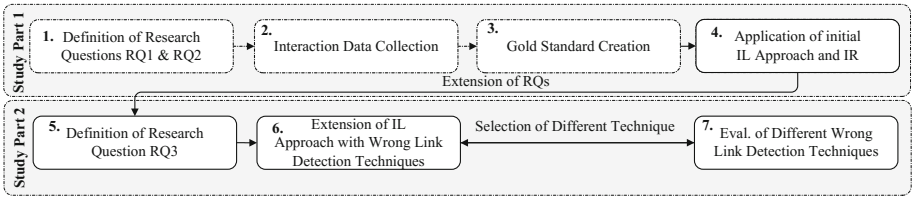

Fig. 2. Experimental design: overview of performed activities

4.1 Research Questions

The initial purpose of this study was to calculate precision and real recall values instead of relative recall as in our last study, for our approach (RQ_1) and for comparison also for IR (RQ_2) [11]. After we realized that the precision of our IL approach was not sufficient for direct usage of the trace links with the data of the student project we investigated the improvement of precision and thus detection techniques for wrong trace links (RQ_3). Thus the research questions we answer in the two parts of our study are:

RQ_1: *What is the precision and recall of IL created trace links?* Our hypothesis was that IL has very good precision and good recall.

[7] http://esprima.org/.

RQ$_2$: *What is the precision and recall of IR created trace links?* Our hypothesis was that IR has bad precision and good recall.

RQ$_3$: *What is the precision and recall of IL with detection techniques for wrong trace links?* Our hypothesis was that detection techniques utilizing details of the interaction log like the time stamp, and detection techniques considering the source code like using the source code structure should enhance precision considerably and keep reasonable recall.

4.2 Gold Standard Creation

The left side of Table 1 shows the overview of all recorded interactions for user stories or sub-tasks and the number of involved source code files of the three developers which we used for further processing and evaluation in our study.

Table 1. Interaction data and gold standard

	Interaction logs			Gold standard creation					
	#Req.	#Interactions	#Code. Files	#Req.	#Code. Files	#Link Cand.	#Rated Correct	#Rated Wrong	#Rated Unknown
Dev_1	12	628.502	155	3	99	139	37	90	2
Dev_2	20	506.726	273	11	141	374	128	241	5
Dev_3	16	893.390	256	5	83	189	52	123	14
Sum	42[a]	2.028.618	312[a]	19	151[a]	692	217	454	21

[a]Same issues and source files used by different developers have been accumulated

The right side of Table 1 shows the overview of the gold standard. For the gold standard creation we first selected 21 user stories of the 42 requirements, since these 21 user stories were assigned directly to the three developers. The others had been assigned to other developers or had a different issue type. Through this we made sure that the developers knew the requirements very well. We further excluded two of the 21 user stories. For one user story one developer had not stopped the interaction recording and thus links to almost all source code files in the Git repository had been created. The other user story was the first in the interaction logs of a developer and no activation event was recorded for that user story.

To limit the link candidates to a reasonable amount we considered all possible link candidates between user stories and code files tagged with the same feature. For the remaining 19 user stories we selected all code files from the Git repository with the same feature tag (cf. Sect. 2.2). This excluded in particular files with a format different than *javascript and json and xml*. Examples for such files are html files and build scripts. After this 151 code files, as shown in the sixth column of the last row of Table 1, remained. Then we created all possible link candidates between user stories and code files with the same tag. This resulted in 692 link candidates.

We provided a personalized questionnaire with link candidates for the three developers. The developers labeled the links as correct (217), wrong (454)

or unknown (21). The latter means they did not have the competence to judge. The developers also confirmed that all feature labels were correct. The three developers worked on their personalized questionnaire in individual sessions lasting between two to three hours in a separate office room in our department and had the possibility to ask questions if something was unclear. Thus initially all links of the gold standard were only rated by one developer. After the first part of our study we checked the link ratings of the developers for plausibility. By inspecting the source code files and requirements involved in each link we manually checked 113 wrong links created by our approach.

4.3 Part 1: Trace Link Creation with IL and IR

We initially created trace links with our IL approach (cf. Sect. 3) and with the common IR methods vector space model (VSM) and latent semantic indexing (LSI) [2,4]. We applied both approaches to the user stories together with their sub-tasks (see Sect. 2.2) and to the 151 code files used for the gold standard creation. We only used these code files, as we only had the gold standard links for them.

For IL we combined the interactions of a user story with the interactions of the corresponding sub-task for further evaluations, as the sub-tasks describe details for implementing the user story. From the resulting link candidates we removed all links to code files not included in the gold standard. We applied IR to the texts of user stories and corresponding sub-tasks and to the 151 code files used for the gold standard. In addition we performed all common IR preprocessing steps [1,2], i.e. stop word removal, punctuation character removal and stemming. We also performed camel case identifier splitting (e.g. *PatientForm* becomes *Patient Form*), since this notation has been used in the source code [4]. Since the user stories contained only very short texts, the used threshold values for the IR methods had to be set very low.

4.4 Part 2: Detection Techniques for Wrong Trace Links

Since our IL approach had worse precision values as we expected, we decided to investigate how IL can be extended by the detection of wrong trace links. Thus we extended our initial study with a second part in which we wanted to answer RQ_3 (cf. Sect. 4.1) for the evaluation of wrong link detection techniques. We looked at two different kind of wrong trace link detection techniques. The first set of techniques was based on the data available in the interaction logs. The second set of techniques used the source code files touched by interactions and data around these files. The main idea was to directly detect link candidates not relevant for a user story or code files not relevant for a user story.

For the interactions logs we used (a) the type of interaction, i.e. whether an interaction is a select or a edit, (b) the duration of interactions based on the logged time stamp and (c) the frequency how often an interaction with a source code file occurred for a user story. The rationale was that (a) edit events are more likely than select events to identify code necessary for a user story and

that (b, c) a longer duration of the interaction or higher frequency signify that the developer made a more comprehensive change and not only a short edit e.g. correcting a typo noticed when looking at a file.

For source code we used (a) the ownership that is the developer who created the interaction, as one developer might have worked less disciplined than others (b) the number how often source code files were interacted with for different user stories, as files used for different user stories might be base files which had not been considered relevant for the gold standard by the developers (c) filtering on only JavaScript source code files as other formats might not be so relevant for a user story and (d) the code structure for the source code files involved in one user story to detect files which had no relation in the code structure to other files, as the unrelated code files might signify a different purpose than the user story. We then combined the most promising techniques. Altogether we implemented wrong link detection so that link candidates were removed when their logged values were below a certain threshold, different of a certain type or when the source code file did not match the aforementioned criteria. We choose the threshold, the type and the combination of thresholds and source code filter criteria to optimize the precision of the links created by IL and minimize the effect on the recall.

5 Results

This section reports the results of evaluations along with answering the RQs.

5.1 Part 1: Precision and Recall for the Initial Evaluation

Table 2 gives an overview of the evaluations performed as described in Sect. 4.3. Our approach created 372 link candidates, 212 of them were wrong. 57 correct links were not found. We can answer RQ_1 as follows: The precision for our IL approach is 43.0% and recall is 73.7%

Table 2. Precision and recall for IL and IR

Approach	GS links	Link Cand.	Correct links	Wrong links	Not found	Precision	Recall	$F_{0.5}$	F_1
IL	217	372	160	212	57	0.430	0.737	0.469	0.543
$IR_{VSM(0.3)}$	217	191	38	153	179	0.199	0.175	0.194	0.186
$IR_{VSM(0.2)}$	217	642	104	538	113	0.162	0.480	0.187	0.242
$IR_{LSI(0.1)}$	217	102	35	67	182	0.343	0.161	0.280	0.219
$IR_{LSI(0.05)}$	217	363	77	286	140	0.212	0.355	0.231	0.266

We can answer RQ_2 looking at the different IR variants with different thresholds: with very low thresholds the best achievable precision is 34.3% (LSI(0.1))

Table 3. Duration based IL improvement

Dur.	GS	Link Cand.		Correct		Wrong		Not	Precision		Recall		$F_{0.5}$	F_1
(sec)	links	All	Edit	All	Edit	All	Edit	found	All	Edit	All	Edit		
1	217	372	220	160	107	212	113	57	0.430	0.486	0.737	0.493	0.488	0.490
10	217	317	199	144	104	173	95	73	0.454	0.523	0.664	0.479	0.513	0.500
60	217	231	167	113	90	118	77	104	0.489	0.539	0.521	0.415	0.508	0.469
180	217	183	142	93	78	90	64	124	0.508	0.549	0.429	0.359	0.497	0.435
300	217	154	122	81	70	73	52	136	0.526	0.574	0.373	0.323	0.496	0.413

and the best achievable recall 48.0% (VSM(0.2)). These results are bad compared to IL and bad compared to typical IR-results on structured data [9] (cf. Sect. 2.1).

As the IL precision was much lower than expected, we investigated whether there was a problem with the gold standard. We therefore checked manually 113 wrong links which resulted from edit interactions (see next section) and confirmed that these links are really wrong. We concluded that the developers had not used the interaction logging properly and worked on code not relevant for the activated user story. This happened typically for smaller code changes on the fly beside the implementation of the activated user story. So for example developers updated a file from which they had copied some code, but they did not activate the requirement the change should have been associated with.

5.2 Part 2: Precision and Recall Using Wrong Link Detection

In this section we report on the answers to RQ_3. Table 3 shows the results for focusing on edit interactions and different minimal duration. The first row corresponds to our IL approach without any restrictions. It shows that by focusing on edit interactions the precision slightly improves from 43.0% to 48.6%. As focus on edit always improved the precision a little, we only report the F-measure for IL focused on edits and we only describe these numbers in the following text. When increasing the minimum duration for an interaction precision can be

Table 4. Frequency based IL improvement

Frequency	GS	Link Cand.		Correct		Wrong		Not	Precision		Recall		$F_{0.5}$	F_1
	links	All	Edit	All	Edit	All	Edit	found	All	Edit	All	Edit		
1	217	372	220	160	107	212	113	57	0.430	0.486	0.737	0.493	0.488	0.490
2	217	314	220	142	107	172	113	75	0.452	0.486	0.654	0.493	0.488	0.490
5	217	220	191	113	98	107	93	104	0.514	0.513	0.521	0.452	0.499	0.480
10	217	181	169	99	93	82	76	118	0.547	0.550	0.456	0.429	0.521	0.482
20	217	158	151	90	87	68	64	127	0.570	0.576	0.415	0.401	0.530	0.473
100	217	86	86	59	59	27	27	158	0.686	0.686	0.272	0.272	0.526	0.389

improved up to 57.4%. This impairs of course recall. We show at the end of this section how recall can be improved by using the code structure.

Table 4 shows the results for different minimal frequencies within one interaction log. Again row one gives the numbers for the original approach. Here the improvement is stronger leading to a precision of 68.6% for a frequency of 100. In particular, by this restriction all select interactions are removed. However, recall is even more impaired.

Table 5. Developer specific differences

Developer	GS	Link Cand.		Correct		Wrong		Not	Precision		Recall		$F_{0.5}$	F_1
	links	All	Edit	All	Edit	All	Edit	found	All	Edit	All	Edit		
Dev_1	37	41	17	19	6	22	11	18	0.463	0.353	0.514	0.162	0.286	0.222
Dev_2	128	252	155	110	79	142	76	18	0.437	0.510	0.859	0.617	0.528	0.558
Dev_3	52	77	46	30	21	47	25	22	0.390	0.457	0.577	0.404	0.445	0.429

Table 5 shows the distribution for the three developers. One can see that developer Dev_2 was the most active and Dev_3 contributed more than Dev_1. However, for all three the interactions led to more wrong than correct links. So precision does not differ much.

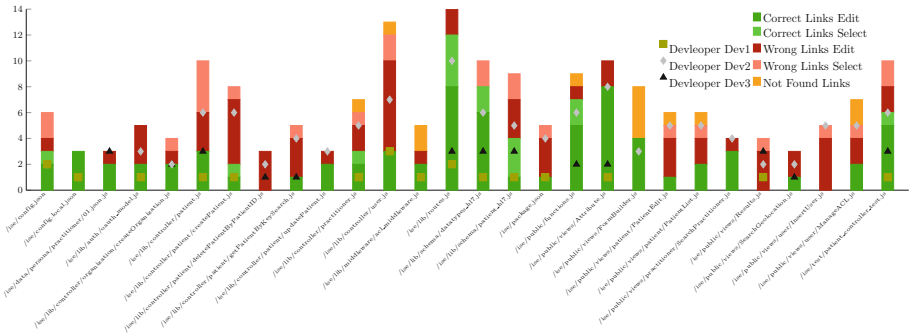

Fig. 3. Code files which had interactions in 3 or more user stories

Figure 3 shows the 28 code files which have been touched in interactions for three or more user stories. Furthermore it shows how often each developer touched these files. The developer distribution shows that some of the files have been touched by different user stories from one developer and some from several developers. One can see that only three out of 28 files have only wrong link candidates. Also files which have many link candidates sometimes have many correct link candidates and sometimes not. So there is no clear pattern that these files are the reason for more wrong link candidates.

Table 6. Source code based improvements

Code Res.	GS links	Link Cand. All	Link Cand. Edit	Correct All	Correct Edit	Wrong All	Wrong Edit	Not found	Precision All	Precision Edit	Recall All	Recall Edit	$F_{0.5}$	F_1
none	217	372	220	160	107	212	113	57	0.430	0.486	0.737	0.493	0.488	0.490
>3 US	217	208	92	83	43	125	49	134	0.399	0.467	0.382	0.198	0.368	0.278
Only .js	186	327	203	129	99	198	104	57	0.394	0.488	0.694	0.532	0.496	0.509
Con.	217	274	169	147	99	127	70	70	0.536	0.586	0.677	0.456	0.554	0.513

This is confirmed in Table 6 which shows the results for the different source code restrictions with the first row showing the numbers without restrictions. The second row shows the precision for code which was touched by interactions in three or more user stories. Here the precision increased slightly to 46.7%. The third row shows a precision 48.8% when only looking at Javascript files. The best precision of 58.6% could be achieved when removing code files which were not connected by source code relations to other code files of the same user story.

When looking at the individual techniques for detecting wrong links we thus can answer RQ_3 as follows: The best precision 68.6% can be achieved with a minimum frequency of 100. This leads to a recall of 27.2%. The second best precision 58.2% can be achieved with removing files which are not connected. This leads to a recall of 45.6%.

Table 7. Combination of improvements

Code Con.	*Freq.*	*Code Struct*	GS links	Link Cand. All	Link Cand. Edit	Correct All	Correct Edit	Wrong All	Wrong Edit	Not found	Precision All	Precision Edit	Recall All	Recall Edit	$F_{0.5}$	F_1
True	*20*	*0*	217	124	123	82	82	42	41	135	0.661	0.667	0.378	0.378	0.578	0.482
True	*20*	*4*	217	151	148	101	101	50	47	116	0.669	0.682	0.465	0.465	0.624	0.553
True	*100*	*0*	217	71	71	47	47	24	24	170	0.662	0.662	0.217	0.217	0.469	0.326
True	*100*	*4*	217	87	87	58	58	29	29	159	0.667	0.667	0.267	0.267	0.513	0.382

We therefore also investigated in the combination of these two techniques. We first removed the not connected code files and then restricted the remaining interaction links wrt. frequency. Table 7 shows the resulting precision of 66.7% for frequency 20 ($F_{0.5}$ is 0.578) and 66.2% for frequency 100 ($F_{0.5}$ is 0.469).

So for frequency 100 precision decreased when looking at connected files. For frequency 20 we get the best $F_{0.5}$-measure of all evaluations. We applied the recall improvement (IL_i) to both settings. Again frequency 20 yielded the best results.

Altogether RQ_3 can be answered as follows: with the wrong link detection techniques we could improve precision from 43.0% up to 68.2% (increase of 25.2%). The recall decreased from 73.7% without wrong link detection to 46.5%. This yields the best $F_{0.5}$-measure of 0.624.

5.3 Discussion

In the following we discuss all of our hypotheses wrt. IL and the rationale for the detection techniques. The bad precision compared to our previous study for IL clearly indicates that the developers did not use the recording in a disciplined way. The detailed evaluations for the developers did not show big differences, so this was true for all three developers. We tried several detection techniques for wrong links: Focus on edit interactions, duration, source code owner, source code type and removing of files with many links did not yield considerable precision improvement. Only frequency and removal of non-connected files improved the precision considerably up to almost 70% with recall above 45%. (cf. Sect. 2.1). For our purpose they are not sufficient, as this still means that our approach would create thirty percent links not directly usable for the developers.

We thus see three further directions of research. (a) We can try to come up with further techniques to detect wrong links which yield a precision close to 100%. (b) We can try to support the developers in applying interaction recording in a more disciplined way. The results of our previous paper [11] on the Mylyn project showed that it is possible for developers to use interaction recording in a disciplined way. It could be that students are particularly bad with this discipline. (c) Instead of automatic link creation support we can generate links through IL as recommendations to the developers.

In previous research [6] we had used more coarse-grained VCS change logs to create links and had given the developers different means to create links based on the logs during a sprint or at the end of a project. We could use our IL approach to give recommendations to the developers at different points in the sprint or project which links to create based on their interactions. Then developers have to detect the wrong links themselves. However, we would like to avoid such overhead for the developers as much as possible.

6 Threats to Validity

In this section we discuss the threats to validity of our study. 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. Wrt. 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. Thus, 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. In the Mylyn open source project, used in our last study, the developers used their own implemented interaction logging approach and thus worked very disciplined. It is very likely that the student developers did not apply the interaction logging as disciplined as the Mylyn developers, since they had no awareness for it. Interaction recording is not yet applied often in industry. So it is an open question how disciplined interaction logging can be achieved.

7 Related Work

In our previous paper [11] we discuss other work on IR and interaction logging such as the systematic literature review of Borg on IR trace link creation [2] or Konopkas approach [12] to derive links between code through interaction logs. Most similar to our work is the approach of Omoronyia et al. [20] who capture interactions between source code and structured requirements specified as use cases. We adopt their approach using select and edit events for trace link creation. In contrast to our goal their tool support focuses on visualizing the trace links after a task has been performed and not on direct availability and usage of trace links.

For this paper most relevant is research on the quality of recorded interaction. We only found a very recent study of Soh et al. [22] studying interactions recorded in Mylyn. They show that the assumptions that the time recorded for an interaction is the time spent on a task and that an edit event recorded by Mylyn corresponds to modification in the code are not true. They could detect these differences by comparing the interactions and videos capturing developer behavior in a quasi-experiment. These differences are not due to any misbehavior of the developers, but only due to Mylyns recording algorithm. For example searching and scrolling is not counted in the time spent and idle time is not treated correctly. In this study these problems do not apply as we used a different logging environment. We are not aware of any noise problems with this environment. Similar to their work, we also use duration as an indicator for a relevant event.

8 Conclusion and Outlook

In this paper we investigated the precision and recall of our IL-approach for trace link creation in a student project. Contrary to our previous work the original approach only achieved a precision of about 50%. We therefore implemented several techniques for the detection of wrong links: Focus on edit interactions, duration, source code owner, source code type and removing of files with many links did not yield considerable precision improvement. Only frequency and removal of non-connected files improved the precision considerably up to almost 70% with above 45% recall. As discussed in Sect. 5.3 this is not sufficient for our purpose.

We are starting to apply the IL-approach in another student project. In this project we will make sure through regular inspections that the students apply the approach in a disciplined way. We will use the two best improvement techniques as quick indicators for undisciplined usage and interview the students for reasons of such usage. Given sufficient precision we plan to also create the links immediately after each interaction and observe the use of the links in the project.

Acknowledgment. We thank the students of the project for the effort.

References

1. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval, 2nd edn. Pearson/Addison-Wesley, Harlow, Munich (2011)
2. Borg, M., Runeson, P., Ardö, A.: Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. Empir. Softw. Eng. **19**(6), 1–52 (2013)
3. Briand, L., Falessi, D., Nejati, S., Sabetzadeh, M., Yue, T.: Traceability and SysML design slices to support safety inspections. ACM ToSEM **23**(1), 1–43 (2014)
4. De Lucia, A., Di Penta, M., Oliveto, R.: Improving source code lexicon via traceability and information retrieval. IEEE TSE **37**(2), 205–227 (2011)
5. De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Recovering traceability links in software artifact management systems using information retrieval methods. ACM ToSEM **16**(4), 1–50 (2007)
6. Delater, A., Paech, B.: Tracing requirements and source code during software development: an empirical study. In: International Symposium on Empirical Software Engineering and Measurement, Baltimore, MD, USA, pp. 25–34. IEEE/ACM, October 2013
7. Falessi, D., Di Penta, M., Canfora, G., Cantone, G.: Estimating the number of remaining links in traceability recovery. Empir. Softw. Eng. **22**(3), 996–1027 (2016)
8. Gotel, O., Cleland-Huang, J., Hayes, J.H., Zisman, A., Egyed, A., Grunbacher, P., Antoniol, G.: The quest for ubiquity: a roadmap for software and systems traceability research. In: RE Conference, pp. 71–80. IEEE, September 2012
9. Hayes, J., Dekhtyar, A., Sundaram, S.: Advancing candidate link generation for requirements tracing: the study of methods. IEEE TSE **32**(1), 4–19 (2006)
10. Hübner, P.: Quality improvements for trace links between source code and requirements. In: REFSQ Workshops, Doctoral Symposium, Research Method Track, and Poster Track, Gothenburg, Sweden, vol. 1564. CEUR-WS (2016)
11. 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
12. Konopka, M., Navrat, P., Bielikova, M.: Poster: discovering code dependencies by harnessing developer's activity. In: ICSE, pp. 801–802. IEEE/ACM, May 2015
13. Kuang, H., Nie, J., Hu, H., Rempel, P., Lü, J., Egyed, A., Mäder, P.: Analyzing closeness of code dependencies for improving IR-based traceability recovery. In: SANER, pp. 68–78. IEEE, February 2017

14. Kuang, H., Mäder, P., Hu, H., Ghabi, A., Huang, L., Lü, J., Egyed, A.: Can method data dependencies support the assessment of traceability between requirements and source code? J. Softw. Evol. Process **27**(11), 838–866 (2015)
15. Maalej, W., Kurtanovic, Z., Felfernig, A.: What stakeholders need to know about requirements. In: EmpiRE, pp. 64–71. IEEE, August 2014
16. Mäder, P., Egyed, A.: Do developers benefit from requirements traceability when evolving and maintaining a software system? Empir. Softw. Eng. **20**(2), 413–441 (2015)
17. Merten, T., Falisy, M., Hübner, P., Quirchmayr, T., Bürsner, S., Paech, B.: Software feature request detection in issue tracking systems. In: RE Conference. IEEE, September 2016
18. 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
19. Niu, N., Mahmoud, A.: Enhancing candidate link generation for requirements tracing: the cluster hypothesis revisited. In: RE Conference, pp. 81–90. IEEE, September 2012
20. Omoronyia, I., Sindre, G., Roper, M., Ferguson, J., Wood, M.: Use case to source code traceability: the developer navigation view point. In: RE Conference, Los Alamitos, CA, USA, pp. 237–242. IEEE, August 2009
21. Seiler, M., Paech, B.: Using tags to support feature management across issue tracking systems and version control systems. In: Grünbacher, P., Perini, A. (eds.) REFSQ 2017. LNCS, vol. 10153, pp. 174–180. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-54045-0_13
22. Soh, Z., Khomh, F., Guéhéneuc, Y.G., Antoniol, G.: Noise in Mylyn interaction traces and its impact on developers and recommendation systems. Empir. Softw. Eng. 1–48 (2017). https://doi.org/10.1007/s10664-017-9529-x