Using Interaction Data for Continuous Creation of Trace Links between Source Code and Requirements in Issue Tracking Systems

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]** Information retrieval (IR) trace link creation approaches have insufficient precision and do not perform well on unstructured data which is typical in issue tracker systems (ITS). **[Question/ problem]** We are interested in understanding how interaction tracking on artifacts can help to improve precision and recall of trace links between requirements specified unstructured in an ITS and source code. **[Principal ideas/ results]** We performed a study with open source project data in which artifact interactions while working on requirements specified in an ITS have been recorded. **[Contribution]** The results of our study show that precision of interaction-based links is 100% and recall is 93% for the first and 80% for the second evaluated data set relative to IR-created links. Along with the study we developed an approach based on standard tools to automatically create trace links using interactions which also takes into account source code structure. The approach and the study show that trace links creation in practice can be supported with little extra effort for the developers.

Keywords: traceability, continuous, interaction, requirement, source code

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 focus on links between requirements [5]. It is known that precision of IR created links is often not satisfying [14] 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 [15, 29, 12].

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 [6]. 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 [22]. For open source projects ITS are even the de facto standard for

all requirements management activities [26]. 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 [27]. 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 [23]. 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 desirable. Therefore, a trace link creation approach for links between unstructured requirements and code is needed with good precision and recall. It is the goal of our research to develop such an approach [16] based on interaction logs and code relations. Interaction logs capture the source code artifacts touched while a developer works on an issue. We already provided a trace link creation approach based on version control system (VCS) change logs [11]. Interaction logs provide more fine-grained interaction data than VCS change logs. Code relations such as references between classes provide additional information. In this paper we explore the potential of interaction logs and code relations aiming at 100% precision.

To facilitate the usage of such fine-grained interaction logs we provide a trace link creation approach which we call *interaction link* (IL). We study the precision and recall of our approach in comparison with IR created trace links. The overall research question which we answer with our study is

Is there a difference between the application of IR and IL based trace link creation regarding precision and relative recall?

Since it is not possible to get a project from industry or open source which provides both, fine-grained interaction logs and a gold standard for trace links, we do not look at precision and recall of IR and IL wrt. a gold standard. Instead we directly evaluate the precision of IL and IR. Furthermore, we compute the relative recall of IR and IL. Relative recall compares the correct links found by one approach with the correct links found by both trace link creation approaches [13]. This kind of recall is well established in domains in which a gold standard creation and thus absolute recall calculation is not possible, e.g. in the field of search engine comparison [20].

For our study we use the interaction log data, requirements and source code from the Mylyn[1] development project. This interaction log data has also been used by others for different research purposes [21, 18].

The results of the study show that IL has 100% precision and is better than the precision of IR. In addition we show that IL with code relations has also better relative recall than IR. The remainder of this paper is structured as follows. Section 2 gives a short introduction into IR, the creation of trace links, ITS as data source for requirements, the evaluation of trace link creation approaches

[1] `http://www.eclipse.org/mylyn`

and interaction tracking. In Section 3 we discuss related work. Section 4 introduces our trace link creation approach. Section 5 states the research questions which are derived from the general research question introduced above and introduces the experimental design along with the selection of data sets for our study. In Section 6 we present the results of the study and answer the research questions including a discussion. Section 7 discusses the threats to validity of the study. Section 8 concludes the paper and discusses future work.

2 Background

This section introduces the background of our approach and the study.

2.1 IR and the Creation of Trace Links

IR is the computer based search for information within a set of artifacts. IR algorithms are used to execute search queries aiming to retrieve all relevant artifacts while minimizing the non-relevant artifacts [4]. When using IR for trace link creation the query concerns textual similarity between two artifacts. Textual similarity is determined by calculating the cosine similarity and defining a threshold for the calculated cosine similarity. Cosine similarity measures the similarity between the two term vectors representing the artifacts based on the cosine of the angle between the term vectors by a numerical value between 0 and 1 [5]. 0 indicates no similarity between two artifacts and 1 that two artifacts are identical. In order to define if two artifacts are related with each other and should be linked a threshold value for the cosine similarity is used [7]. Thus, varying this threshold value also varies the number of created trace link candidates.

In our study the artifacts are requirements issues and implementation artifacts. There are different IR algorithms. The most common IR algorithms used for trace link creation are vector space model (VSM) and latent semantic indexing (LSI) [5, 14]. Thus, we used these two IR algorithms for comparison with our new trace link creation approach. The difference between VSM and LSI is that VSM uses a more strict term comparison than LSI. Whereas VSM measures the similarity based on terms, LSI measures the similarity based on concepts, which are high level abstractions of the used terms and can been seen as the topics of the artifacts [4]. Thus LSI enables similarity matches between artifacts which do not contain the exactly same terms.

The preprocessing of artifact is essential for the application of an IR algorithm. Typically, preprocessing consist of several steps. Some of them are fundamental and some are specific to the used data sources. In our study we applied the following common preprocessing steps [24, 4, 5]. First we used stop word removal to remove common words which have no impact on the similarity of artifacts (e.g. for, the, a, etc.). Then we performed stemming with the Porter Stemmer algorithm. And we removed punctuation characters. As a specific step we performed camel case identifier splitting (e.g. BugzillaTask becomes Bugzilla Task). Since camel case notation is common in java source code while

requirements use separate words [9, 1], this splitting can significantly improve the similarity of source code and requirements artifacts.

2.2 ITS as Data Source for Requirements

ITS are a common platform for information exchange in software development projects [22]. Often ITS are used as a central information data source and thus also for the definition and management of requirements. Requirements are described as issues which at least consist of a title and a description. A basic feature of ITS is the discussion functionality of issues so that users can create comments for issues. These comments may contain requirement relevant content, e.g. a feature description.

2.3 Evaluation of IR created Trace Links

Approaches on trace link creation, e.g. as described in the overview papers [5, 14], by default use a gold standard to evaluate and compare the approach. Such a gold standard consists of the set of all correct trace links for a given set of artifacts. The creation of such a gold standard is labor intensive as it is necessary to manually check if trace links exist for each pair of artifacts. Therefore many approaches use data sets which are specifically created for the purpose of evaluation, e.g. within a student project [10]. We also plan to evaluate our approach in a student project where we can create the gold standard in parallel to the project. As a first step we wanted to explore the usefulness of interaction logs on existing data.

There are only few realistic data sets with interaction logs (cf. next Subsection 2.4). The creation of a complete gold standard for such a project is not feasible. Therefore we only evaluate the precision of the links found by IR or IL and we compute the relative recall.

Precision (P) is the amount of correct links (true positives, TP) within all links found by an approach, i.e. the sum of TP and not correct (false positive, FP) links. Recall (R) is the amount of TP links found by an approach within all existing correct links, i.e. the sum of TP and false negative (FN) links:

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

According to [15] values for P and R of IR for structured requirements can be categorized in three quality levels. *Acceptable* values for R are between 60 and 69% and for P between 20 and 29%. *Good* values for R are between 70 and 79% and for P between 30 and 49% and *excellent* values for R are between 80 and 100% and for P between 50 and 100%. Merten et al. [27] reported varying results for using IR on unstructured requirements data from ITS, i.e. they tried to achieve a 100% for R with different IR algorithms and different preprocessing steps. Then their best values for P were up to 11%. Considering other approaches for link creation between code and requirements using open source projects as

data source Ali et al. also used VSM for trace link creation [2] and achieved similar but also very project specific results for P (between 15 and 77%). De Lucia et al. [10] report values of 90% for R and 25% for P for link creation between structured requirements and source code by using LSI in combination with categorization.

To evaluate our trace link creation approach we use the relative recall measure [13] as we do not have a gold standard. Relative recall is used if it is not possible to get all correct values for a data set due to the size of the data set. It is a well-established standard measure in the domain of web search engine performance and quality measuring [20]. Relative recall uses all correct links available as comparison measure for calculating the recall of a single approach. Therefore the relative recall for IL (RR_{IL}) and for IR (RR_{IR}) are defined as:

$$RR_{IR} = \frac{TP_{IR}}{TP_{IR} + TP_{IL}} \qquad RR_{IL} = \frac{TP_{IL}}{TP_{IL} + TP_{IR}}$$

2.4 Interaction Logs and Code Structure

Interaction logs are all developer interactions with artifacts managed by an IDE. Common IDEs like Eclipse[2] provide the functionality to record these interactions [28]. For the development of our approach we used interactions recorded with the Eclipse Mylyn extension during the open source development of Mylyn. Mylyn logs edit, select and other events after a developer has selected an issue from an associated ITS and activated the recording. Interactions for an issue are recorded until the developer finishes working on the issue, e.g. by closing the issue, by switching to another issue or by explicitly stopping the recording of interactions. For the development of Mylyn the developers use Mylyn together with the ITS Bugzilla[3] which is also used for requirements capture. The Mylyn developers are encouraged to trigger recording when they work on the implementation of a requirement. These interaction logs are accessible as attachments of the issues in the Mylyn Bugzilla ITS.

Interaction logs can have different event types e.g. edit and select events triggered by a developer and system generated events like propagation, command and preference. An interaction log entry comprises the event type E touching the implementation artifact I while working on requirement A. Based on such an interaction log entry a trace links can be created between A and I. Interaction logs enable the link creation on class (file), method and attribute granularity level, i.e. all parts of the source code abstract syntax tree (AST) model.

We use the term code structure to denote the following relations between two classes in the Mylyn name space: a class implements an interface, a class extends another class or a class references other classes in its attributes. These relations can be used for link creation as follows: If a trace link ($A \rightarrow X$) from requirement A to class X has been created, for each class Y which is related

[2] `http://www.eclipse.org`
[3] `https://bugs.eclipse.org/bugs/describecomponents.cgi?product=Mylyn`

with X also a trace link to the requirement A is created ($A \rightarrow Y$). These related classes are likely relevant to the implementation of A. Clearly, this can be applied transitively and in consequence it is theoretically possible that from a single class all existing classes are linked. Therefore we explored different nesting levels in our study.

3 Related Work

In the following we first discuss related work on IR-based trace link creation approaches for structured and unstructured requirements and for considering code structure. Then we discuss related work on the usage of interaction logs. Borg et al. [5] present a current overview of IR based trace link creation approaches based on a systematic literature review. 46 of the 79 analyzed approaches deal with trace link creation between source code and requirements. In contrast to our study most of the approaches use laboratory settings (i.e. student projects) instead of real (open source) projects. We used the assessment of IR algorithms presented by Borg to select VSM and LSI as comparison algorithms for our approach. De Lucia et al. [10] is an example for such a study in which the usage of LSI and VSM are compared. As a result this study reports about possible improvements when using LSI. Our study setup is similar to theirs, since we share the research goal of improving trace link creation. McMillan et al. [25] use source code structure information to improve results of trace links created by VSM. We adapt the use of source code structure in our approach. Instead of using the structure only for verification of already created links by IR we use the source code structure to create additional new links.

Merten et al. [26] have evaluated the application of IR-based trace link creation algorithms in ITS and thus on unstructured requirements data. One of their findings was that preprocessing of the unstructured data is essential for reasonable application of IR. Another finding was that it is not possible to achieve good results for both, precision and recall. In our study we create links between (requirements) issues and source code instead of links between issues.

To identify related work using interaction data we completely explored the Mining Software Repositories conference proceedings, but did not find any approaches using interactions for trace link creation. In consequence we also partly searched in the ICSE and RE proceedings and identified the following relevant publications.

Kersten et al. [17] describe the initial version of Mylyn called Mylar. The basic idea of Mylyn is to reduce the information overload in an IDE by exploiting the interactions of a developer. To do so Mylyn provides the functionality to associate interactions to issues from an ITS within the IDE. With our approach we use these interaction logs available in Mylyn by filtering and aggregating the logs on different levels of granularity and directly providing links in the ITS to the code.

Konopka et al. [19] show that interaction logs are helpful to derive links between development artifacts. They also use Mylyn generated interaction logs and data from the Mylyn project for development and evaluation of their approach.

We adopt interaction based link creation for our trace link creation approach, but in contrast to their focus on code relations we derive links between unstructured requirements and source code. Omoronyia et al. [30] capture interactions between source code and structured requirements specified as use cases to infer trace links based on statistical evaluation of the interactions. 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. Asuncion and Taylor [3] describe the principle of recording interaction for trace link creation, but do not provide a tool. In contrast to our approach their focus has been trace link creation between structured requirements and design oriented artifacts. In our earlier work we used VCS based changed logs, a coarse-grained form of interaction logs, using work items as intermediate elements to create trace links between source code and requirements [11]. Our actual approach improves this earlier work. With more fine grained interactions more detailed trace links can be created.

4 Interaction Log Trace Link Creation Approach

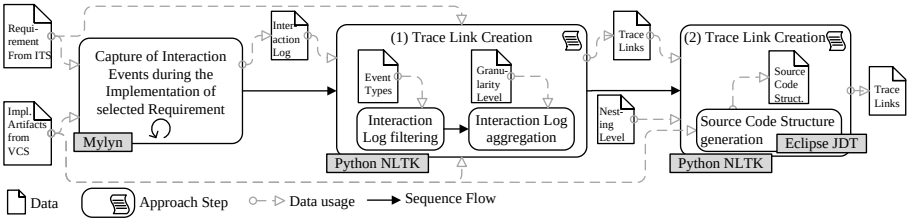

Fig. 1: IL Approach Overview

Figure 1 shows the overview of our IL approach. After the capture of the interaction logs in Mylyn there are two steps to create links. In the first step links are created based solely on interaction logs. In the second step the source code structure is used to create further trace links between requirements and code. We implemented both trace link creation steps in a Python and Java based tool. The NLTK library[4] is used to create the trace links. In the second step Eclipse JDT library[5] is used to create the code structure considered for trace link creation. The used interaction logs are based on the selected requirement in the ITS and the implementation artifacts managed within a VCS. In our approach we only use edit and select events, since these are directly triggered by a developer and indicate relations between the affected artifacts and the processed requirement (filtering). Trace links are created between a requirement and all source code artifacts touched by select or edit interaction events. We support aggregation of links, e.g. if trace links are created to multiple methods of a class, these links are

[4] `http://www.nltk.org/`, Python Natural Language Toolkit
[5] `http://www.eclipse.org/jdt/`, Eclipse Java development tools

aggregated to a single link on file level. In our tool the granularity level of the created trace links is configurable. To be comparable with IR created trace links, which only support file granularity we configure our approach in the study to aggregate interaction log created links to file level. Also the usage of source code structure is configurable wrt. the nesting level of source code relations. E.g. if there are classes *A, B, C* and *D* with the relations $A \rightarrow B \rightarrow C \rightarrow D$ and there is a trace link, created by interaction logs, between requirement R and class A ($R \rightarrow A$), then the nesting level two will result in the creation of two additional trace links $R \rightarrow B$ and $R \rightarrow C$.

5 Experiment Design

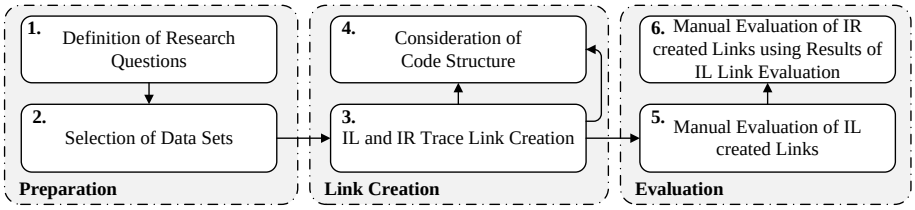

Fig. 2: Experimental Design: Overview of Performed Activities

In this section we describe the design of our evaluation experiment. Figure 2 shows the overview of the activities for the experimental design of our study. It is guided by the detailed research questions stated in the following Section 5.1. In the experiment we evaluate two different data sets both taken from the Mylyn project. The detailed characteristics of the data sets and our process to select the two data sets are described in Section 5.2. For each of the two data sets the experiment steps are:

Link Creation Creation of trace links with our IL approach and the two selected IR algorithms VSM and LSI (cf. 5.3) and consideration of the source code structure. We apply this to both IR and IL.

Evaluation Manual evaluation of trace links created with IL followed by manual evaluation of trace links created with IR. In the evaluation of IR trace links we could use the links already verified in the manual evaluation of IL trace links (cf. 5.4).

5.1 Research Questions

Our overall research question is **Is therea difference between the application of IR and IL based trace link creation regarding precision and relative recall?** We divide this into three sub-questions:

RQ_1: *What is the precision of IR and IL created trace links?* Our hypothesis is that the precision of IL is better than IR, since link creation in IL is based on developers' expert knowledge.

RQ$_2$: *What is the relative recall of IR and IL created trace links?* Our hypothesis is that the relative recall of IL is at least as good as the relative recall of IR. On the one hand IL can find links between artifacts which are not textual similar. On the other hand artifacts found by IR are also covered by interactions.

RQ$_3$: *What is the impact of using code structure?* Our hypothesis is that using the code structure in addition to IL and IR improves the relative recall of both trace link creation approaches.

5.2 Selection of Data Sets

The data sets used in our study consist of data from the Bugzilla ITS for requirements and interaction logs and from the Git VCS for implementation artifacts. For trace link creation with our IL approach we used all three data sources (requirements, implementation artifacts, interaction logs) whereas for IR-based trace link creation only requirements and implementation artifacts have been used. Issues in the Mylyn project have been created starting from early 2005, however the open source development of Mylyn really started at the beginning of 2007 when its source code first was made publicly available. The development activity of Mylyn decreased in the last years but is still ongoing. A reason for this is that the major features are already implemented and development efforts mostly concern bug fixing.

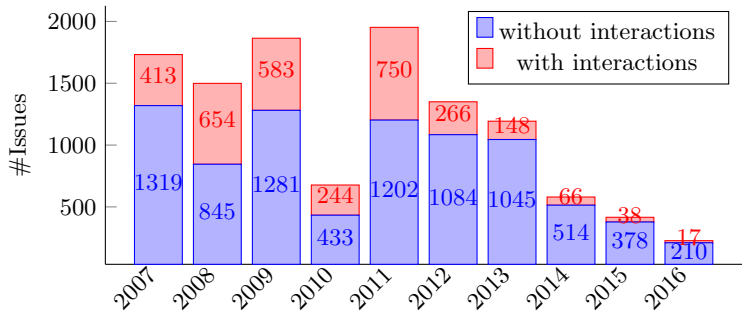

Fig. 3: Issue in the Mylyn Bugzilla ITS per Year

Figure 3 shows the number of issues with and without interaction per year until mid of June 2016 when we fetched the data for our study. Till then there were a total of 11490 issues from which 3179 (27.7%) have interaction logs attached and therefore are suitable for our study. In total the 3179 issues have over 3 million interaction log entries attached. Based on these general data characteristics we decided to evaluate only a subset of the existing interaction logs by selecting a suitable subset of requirements issues. We used the following criteria for the requirements selection:

C$_1$: There should be two distinct data sets from different project phases, i.e. from early phase and later phase. Thereby we want to check whether IL trace link creation is applicable for different project circumstances.

C_2: The number of interactions in the two sets should be as similar as possible to ensure the comparability of the two data sets. Due to the data characteristics this criteria could only be fulfilled up to a certain extent, since also the number of interactions by issue decreases during the years.

These criteria resulted in the creation of two data sets. The *first data set* R_{2007} consists of the first 50 requirements issues in 2007 (and the corresponding interaction log and code) and the *second data set* R_{2012} consists of the first 50 requirements issues in 2012 (and the corresponding interaction logs and code). We used the first requirements of the years, as the Mylyn project employs an annual release cycle with a major release every June. Therefore, new requirements are mostly created at the beginning of a year whereas around the release date more bugs are created. Requirements are described as natural language text using the Bugzilla issue format, i.e. a title, a description text and technical meta-data like the affected components and the assignee. For each data set requirement issues including the comments and interaction logs have been downloaded from Bugzilla. Comments have been included, since they often contain requirement relevant information, e.g. changes to the functionality initially stated in the description. Since there is no explicit classification of the issues as requirement or as bug, we performed this classification for the issues by ourselves. First we fetched an overview list with all issue titles and then manually performed the classification of the issues by reading their title. If classification was not possible by only using the title, we also read the issues description. The two requirements sets have slightly different characteristics. In the first phase of the Mylyn project more complex requirements concerning the basic functionality and in the later phase of the project more requirements concerning small and advanced functionality have been implemented.

To identify the code related to the requirements we used a specific VCS version tag. For each data set we sorted all interaction log entries of the interaction logs in chronological order and then used the first version tag after the last interaction log entry. We assume the so selected VCS version comprises the implementation of the 50 requirements. From these implementation artifacts we removed all artifacts which are not textual and cannot be processed with IR such as pictures or binaries.

Table 1: Study Data Sets Overview

Data Set	**#Requirements**	**#Int. Log Entries**	**VCS Version Tag**	**#Impl. Artifacts** *All*	*Textual*	*Touched by IL*
R_{2007}	50	7687	*R_2_0_RC1*	1103	756	585
R_{2012}	50	1660	*R_3_8_3*	3451	2119	172

Table 1 shows the overview of both data sets. As expected there are much more implementation artifacts in the second (later) data set than in the first data set. In contrast, the amount of interaction log entries, overall and also for each requirement, in the second data set is lower than in the first data set.

Therefore, only a minor part of all implementation artifacts are directly touched by interactions.

5.3 IR-based Trace Link Creation

For IR-based trace link creation we applied both IR algorithms VSM and LSI to the two data sets R_{2007} and R_{2012}. Upfront we applied the preprocessing steps as described in Section 2.1 to all used artifacts. We restricted the trace link candidate generation to links from requirements to implementation artifacts.

Table 2: Thresholds and Number of Candidate Links for IR Algorithms

Thresholds*		0.9	0.8	0.7	0.6	0.5	0.4	0.3
R_{2007}	VSM	0	50	596	2347	6419	13798	24040
	LSI	0	3	8	40	142	354	1058
R_{2012}	VSM	185	2268	6431	12333	22397	39434	64284
	LSI	1	14	86	297	920	2424	6014

* Selected values are highlighted

To determine a reasonable threshold for the IR algorithms we initially used approved threshold values of 0.7 for VSM [8] and 0.3 for LSI [10]. While this worked well for R_{2007}, we had to choose different thresholds for R_{2012}. As can be seen from Table 2, which shows the number of candidate links for different IR thresholds, for the second data set the number of generated links increases very quickly with lowering the threshold. To limit the effort for the verification of the links, we used thresholds with less than 1000 links. Clearly, the results for R_{2012} can only be seen as a first indication and can be improved with lower thresholds.

5.4 Data Evaluation

To evaluate the trace links created with our IL approach we compared the created links from both data sets (R_{2007} and R_{2012}) with links created by IR. We used two settings for trace link creation: one with code structure and one without. We performed the following steps to determine TP and FP for these link sets. We manually verified links created by IL in R_{2007} and R_{2012}. Subsequently, we removed these verified links from the IR trace link candidate sets. This resulted in sets of link candidates only found by IR. We also manually verified these links. Finally, we use the verified IR links and the verified IL links to determine the set of links only found by IL.

6 Results

In the following subsections we answer the research questions of our study and discuss the results. In Section 6.1 we answer RQ_1 and RQ_2 concerning precision and relative recall of IL and IR created trace links. This is followed by the answer to RQ_3 concerning the consideration of code structure in Section 6.2.

6.1 Precision (RQ_1) and Relative Recall (RQ_2)

Table 3: Comparison of IR and IL Trace Link Creation

	R_{2007}			R_{2012}		
	IL	IR ($VSM_{0.7}$)	IR ($LSI_{0.3}$)	IL	IR ($VSM_{0.9}$)	IR ($LSI_{0.5}$)
#Link Cand. (LC)	1148	596	1058	240	185	920
#Impl. $\text{Artifact}_{\text{LC}}$	585	203	384	172	171	444
$\#\text{Requirements}_{\text{LC}}$	50	23	46	37	4	34
#True Positive (TP) Trace Links	1148	204 ($118_{IL}+17_{LSI}+69$)	328 ($184_{IL}+37_{VSM}+107$)	240	25 ($6_{IL}+24_{LSI}+1$)	274 ($41_{IL}+24_{VSM}+250$)
#Impl. $\text{Artifact}_{\text{TP}}$	585	126	200	240	24	169
$\#\text{Requirements}_{\text{TP}}$	50	19	41	172	3	28
#TP Trace Links of all Approaches		1324 ($1148_{IL}+69_{VSM}+107_{LSI}$)			491 ($240_{IL}+1_{VSM}250_{LSI}$)	
Precision	1	0.341	0.310	1	0.135	0.298
Relative Recall	0.867	0.154	0.247	0.418	0.051	0.534

Table 3 shows the overview of the number of created trace link candidates, used implementation artifacts, used requirements, correct trace links, implementation artifacts involved in correct trace links, requirements involved in correct trace links, sum of correct trace links created by all approaches together, precision and relative recall for both data sets. Thus, we can answer our research questions as follows.

RQ_1: What is the precision of IR and IL created trace links? For both data sets all links created with our IL approach were correct (100% precision). For IR precision values vary between 13% and 34% with little difference between VSM and LSI for the standard thresholds in the first data sets and big difference for the higher thresholds in the second data set. Thus, IL clearly outperforms IR. Moreover, IL is independent from setting a threshold and finds more correct links than IR for the R_{2007} data set. Nevertheless, there are also links only discovered by IR in this data set.

For our R_{2012} data set the situation is different due to the smaller number of IL created trace links and much larger amount of used implementation artifacts for IR. Note that not all requirements are involved in interaction links in this set. This is due to the fact that some interactions concerned code outside of the VCS tag (e.g. used framework). LSI finds in total more correct trace links for the second data set than IL. This can be explained by the amount of considered requirements and implementation artifacts, i.e. IL considered 37 requirements and 172 implementation artifacts whereas LSI considered 34 and 444. In comparison with the values achieved by current approaches as discussed in Section 2.3 we can state that the 100% precision of IL in a real world setup is unique. The precision of IR is acceptable for the first and good for the second data set. The values for precision are in the range reported by DeLucia [10] (LSI), Ali [2](VSM) and Merten[27] (LSI, VSM, ITS as data source).

RQ_2: What is the relative recall of IR and IL created trace links? The used setting in our experiment resulted in relative recall rates between 5% and 53% (cf. Table 3) for IR and in relative recall rates of 86% and almost 42% for IL. As expected and reported by others [7, 14], IR creates a lot of false positive trace links even with the moderate threshold setting we used for the second data set in our experiment. The difference in relative recall rates between the R_{2007} and R_{2012} data sets in our IL approach can be explained by the characteristics of the

data sets which resulted in a lower number of interactions for the second R_{2012} data set (cf. Section 5.2 ,Table 1: R_{2007} has 7687 interactions on 756 used implementation artifacts, R_{2012} has 1660 interactions on 2119 used implementation artifacts).

6.2 Using Code Structure (RQ_3)

Table 4: Trace Links for different Code Nesting Levels

Nesting Level	$R_{2007}IL$			$R_{2007}IR$					
	#Link Cand.	#TP Links	Precision	#Link Cand. $VSM_{0.7}$	#Link Cand. $LSI_{0.3}$	#TP Links[a] $VSM_{0.7}$	#TP Links[a] $LSI_{0.3}$	Precision $VSM_{0.7}$	Precision $LSI_{0.3}$
0	1148	1148	1.000	596	1058	120	184	0.201	0.174
1	1446	1446	1.000	858	1718	234	338	0.273	0.197
2	1831	1831	1.000	1108	2181	363	562	0.328	0.258
3	2204	2204	1.000	1382	2706	499	805	0.361	0.297
4	2565	2565	1.000	1624	3214	639	1083	0.393	0.337
5	3027	2854	0.943	1915	3927	781	1349	0.408	0.344
6	3531	3202	0.907	2253	4510	947	1612	0.420	0.357
10	5805	3639	0.627	3374	5488	1258	1779	0.373	0.324

[a] Compared to IL

As mentioned in Section 2.4 the first results concern the setting of an appropriate nesting level. Table 4 shows the differences according to the number of created links and their precision for considering code structure with different nesting levels for the R_{2007} data sets. $R_{2007}IL$ refers to links generated by our IL approach and $R_{2007}IR$ to links generated by IR. Since precision for IL drops when considering a nesting level of code relations greater than four, we used this nesting level for the answer of RQ3. It also can be seen that precision of IR only drops for nesting level 10. As our current focus is to maximize the precision of the IL approach, we choose level 4. Clearly, the results for IR could be improved with higher nesting level. We also performed this analysis for our second data set. Since the results are quite similar, we skip their detailed report here.

Table 5: IR and IL Trace Links Considering Code Structure

	R_{2007}			R_{2012}		
	IL	IR ($VSM_{0.7}$)	IR ($LSI_{0.3}$)	IL	IR ($VSM_{0.9}$)	IR ($LSI_{0.5}$)
#Link Cand. (LC)	2565	1624	3143	1126	458	2766
#Impl. $\text{Artifact}_{\text{LC}}$	627	333	516	363	343	702
$\#\text{Requirements}_{\text{LC}}$	50	23	46	37	4	34
#TP Trace Links of Trace Links	2565	698 $(581_{IL}+63_{LSI}+54)$	1214 $(1010_{IL}+62_{VSM}+142)$	1126	108 $(91_{IL}+17_{LSI}+0)$	784 $(491_{IL}+11_{VSM}+282)$
$\#\text{Impl. Artifact}_{\text{TP}}$	627	229	308	363	73	354
#Requirements (TP)	50	22	41	37	4	35
$\#\text{Trace Links}_{\text{TP}}$ by all Approaches		2761 $(2565_{IL}+54_{VSM}+142_{LSI})$			1408 $(1126_{IL}+0_{VSM}+282_{LSI})$	
Precision	1	0.425	0.386	1	0.236	0.283
Relative Recall	0.929	0.253	0.440	0.800	0.077	0.557

RQ_3: What is the impact of using code structure? As shown in Table 5 for both data sets all links created with our IL approach were also correct (100% precision) when considering code structure. Furthermore, relative recall was increased considerably for the second data set. Comparing Table 3 and 5 we can see that the code structure consideration for IL results in five times more trace links for the second data set and twice as much links for the first data set. This

can be explained by the more complex code structure due to the maturity of the project in the second data set.

Both IL and IR considered about 1/3 more implementation artifacts when using code structure. For VSM and LSI in the R_{2007} data set this resulted in an increase of precision and relative recall. This is also true for the R_{2012} data set, except for the precision value of LSI which slightly drops.

In our experiments we could reduce the number of links only found by IR to almost zero by increasing the nesting levels of code relations. However, this also resulted in false positive links for IL which is contrary to our research goal to create trace links with 100% precision. Altogether, we can see that by using code structure we could achieve our research goal of 100% precision and excellent relative recall and that IL outperforms IR for both Mylyn project data sets.

7 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 was only performed by one researcher. However, this researcher is very familiar with the Mylyn project in general, its source code, the used development infrastructure, and has Mylyn specific development experience for almost ten years.

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 higher threshold for the second data set and the nesting level restriction impairs the results for IR. Thus, further comparison of IL and IR for data sets with few interactions is necessary.

Clearly, the external validity depends on the availability of interaction logs and respective tooling and usage of the tooling by developers. Up to now we have only studied one open source project retrospectively. While the generalizability based on one project is clearly limited, we think that using an open source project is not a limitation: Since IL performed quite well in the loosely organized and structured open source project, we expect even better results when applying the approach to a more strictly structured industry project.

8 Conclusion

The results for our IL approach are encouraging. With IL we could create trace links with 100% precision for two different data sets. Also our calculated relative recall values are excellent, i.e. almost 96% for the first and 80% for the second data set. Thus, the approach and the study show that trace link creation in practice can be supported with little extra effort for the developers. Clearly, the comparison with IR is only preliminary. We did not use the common thresholds for the second data set and we could only compute relative recall.

We already created a tool to assess created trace links in detail. The tool enables the automation of all steps necessary to compare two trace link sets on

the basis of single requirements. The usage of this tool for detailed trace link evaluation and the determination of absolute recall values are part of our planed follow up study.

We will investigate the application of our approach including the evaluation of its practicability in a real project. The project started in Fall 2016 and lasts until Spring 2017. In this project we evaluate IL in a different context (Scrum, IntelliJ, Jira) where we can create both, interaction logs and a gold standard, and thus compute recall and provide a full comparison with IR. Furthermore, we evaluate the usage of IL created trace links by incorporating them into the ITS. To improve our approach further, we will investigate the use of existing trace links in combination with IL[16].

Acknowledgment We thank the open source community for providing the data for our research.

References

1. Ali, N., Gueheneuc, Y.G., Antoniol, G.: Requirements Traceability for Object Oriented Systems by Partitioning Source Code. In: Conference on Reverse Engineering. pp. 45–54. IEEE (oct 2011)
2. Ali, N., Gueheneuc, Y.G., Antoniol, G.: Trustrace: Mining Software Repositories to Improve the Accuracy of Requirement Traceability Links. IEEE TSE 39(5), 725–741 (may 2013)
3. Asuncion, H.U., Taylor, R.N.: Automated Techniques for Capturing Custom Traceability Links Across Heterogeneous Artifacts. In: Software and Systems Traceability, pp. 129–146. Springer, London (2012)
4. Baeza-Yates, R., Ribeiro, B.d.A.N.: Modern Information Retrieval. Pearson Addison-Wesley, Harlow, Munich, 2. edn. (2011)
5. Borg, M., Runeson, P., Ardö, A.: Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. Empirical Software Engineering 19(6), 1–52 (may 2013)
6. 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 (feb 2014)
7. Cleland-Huang, J., Berenbach, B., Clark, S., Settimi, R., Romanova, E.: Best Practices for Automated Traceability. Computer 40(6), 27–35 (jun 2007)
8. De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Enhancing an artefact management system with traceability recovery features. In: ICSM. pp. 306–315. IEEE (2004)
9. De Lucia, A., Di Penta, M., Oliveto, R.: Improving Source Code Lexicon via Traceability and Information Retrieval. IEEE TSE 37(2), 205–227 (mar 2011)
10. 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)
11. Delater, A., Paech, B.: Tracing Requirements and Source Code during Software Development: An Empirical Study. In: Int. Symp. on Empirical Software Engineering and Measurement. pp. 25–34. IEEE/ACM, Baltimore, MD, USA (oct 2013)
12. Falessi, D., Di Penta, M., Canfora, G., Cantone, G.: Estimating the number of remaining links in traceability recovery. Empirical Software Engineering (oct 2016)

13. Fricke, M.: Measuring recall. Journal of Information Science 24(6), 409–417 (1998)
14. 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. pp. 71–80. IEEE (sep 2012)
15. Hayes, J., Dekhtyar, A., Sundaram, S.: Advancing candidate link generation for requirements tracing: the study of methods. IEEE TSE 32(1), 4–19 (jan 2006)
16. Hübner, P.: Quality Improvements for Trace Links between Source Code and Requirements. In: Joint Proc. of REFSQ Workshops, Doctoral Symp., Research Method Track, and Poster Track. CEUR-WS, Gothenburg, Sweden (2016)
17. Kersten, M., Murphy, G.C.: Using task context to improve programmer productivity. In: Proceedings of the 14th ACM SIGSOFT int. symp. on Foundations of Software Engineering - SIGSOFT '06/FSE-14. pp. 1–11. ACM, New York, New York, USA (nov 2006)
18. Konopka, M., Navrat, P.: Untangling Development Tasks with Software Developer's Activity. In: Int. Workshop on Context for Software Development. pp. 13–14. IEEE/ACM (may 2015)
19. Konopka, M., Navrat, P., Bielikova, M.: Poster: Discovering Code Dependencies by Harnessing Developer's Activity. In: ICSE. pp. 801–802. IEEE/ACM (may 2015)
20. Kumar, B., Prakash, J.: Precision and relative recall of search engines: A comparative study of Google and Yahoo. Singapore Journal of Library & Information Management 38(1), 124–137 (2009)
21. Maalej, W., Ellmann, M.: On the Similarity of Task Contexts. In: Int. Workshop on Context for Software Development. pp. 8–12. IEEE/ACM (may 2015)
22. Maalej, W., Kurtanovic, Z., Felfernig, A.: What stakeholders need to know about requirements. In: EmpiRE. pp. 64–71. IEEE (aug 2014)
23. Mäder, P., Egyed, A.: Do developers benefit from requirements traceability when evolving and maintaining a software system? Empirical Software Engineering 20(2), 413–441 (apr 2015)
24. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to information retrieval. Cambridge Univ. Press, Cambridge UK, 1. publ. edn. (2008)
25. McMillan, C., Poshyvanyk, D., Revelle, M.: Combining textual and structural analysis of software artifacts for traceability link recovery. In: ICSE Workshop on Traceability in Emerging Forms of SE. pp. 41–48. IEEE (may 2009)
26. Merten, T., Falisy, M., Hübner, P., Quirchmayr, T., Bürsner, S., Paech, B.: Software Feature Request Detection in Issue Tracking Systems. In: RE. IEEE (sep 2016)
27. 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: REFSQ. vol. LNCS 9619, pp. 45–62. Springer, Gothenburg, Sweden (2016)
28. Murphy, G., Kersten, M., Findlater, L.: How are Java software developers using the Elipse IDE? IEEE Software 23(4), 76–83 (jul 2006)
29. Niu, N., Mahmoud, A.: Enhancing Candidate Link Generation for Requirements Tracing: The Cluster Hypothesis Revisited. In: RE. pp. 81–90. IEEE (sep 2012)
30. Omoronyia, I., Sindre, G., Roper, M., Ferguson, J., Wood, M.: Use Case to Source Code Traceability: The Developer Navigation View Point. In: RE. pp. 237–242. IEEE, Los Alamitos, CA, USA (aug 2009)