Empirical Software Engineering (2020) 25:4350–4377
https://doi.org/10.1007/s10664-020-09831-w

Interaction-based creation and maintenance of continuously usable trace links between requirements and source code

Paul Hübner[1] · **Barbara Paech**[1]

Published online: 11 August 2020

Abstract

Trace links between requirements and code are beneficial for many software engineering tasks such as maintenance, program comprehension, and re-engineering. If trace links are created and used continuously during a project, they need to have high precision and recall to be useful. However, manual trace link creation is cumbersome and existing automatic trace link creation methods are typically only applied retrospectively and to structured requirements. Therefore, they focus on recall and accept manual effort to cope with low precision. Such manual effort is not acceptable continuously. Furthermore, the maintenance of existing links along with changing artefacts in a project is neglected in most automatic trace link creation approaches. Therefore, we developed and evaluated an interaction log-based trace link creation approach *IL* to continuously provide correct trace links during a project. *IL* links unstructured requirements specified in an issue tracker and source code managed in a version control system. In the latest version, IL_{Com}, our approach uses the interactions of developers with files in an integrated development environment and issue identifiers provided in commit messages to create trace links continuously after each commit. In this paper, we present IL_{Com}, its most recent evaluation study, and a systematic literature review (SLR) about trace link maintenance (TM). We also present a TM process for IL_{Com} based on two approaches from our SLR. In the evaluation study, we show that precision of IL_{Com} created links is above 90% and recall almost at 80%. In the SLR, we discuss 16 approaches. Our approach is the first trace link creation approach with very good precision and recall and integrated trace maintenance.

Keywords Traceability · Interaction · Maintenance · Source code · Requirement

This article belongs to the Topical Collection: *Requirements Engineering for Software Quality (REFSQ)*

Communicated by: Eric Knauss, Michael Goedicke and Paul Grünbacher

✉ Paul Hübner
huebner@informatik.uni-heidelberg.de

Barbara Paech
paech@informatik.uni-heidelberg.de

[1] Institute for Computer Science, University of Heidelberg, Heidelberg, Germany

1 Introduction

Existing trace link creation approaches between requirements and code are most frequently based on information retrieval (IR) and on structured requirements, such as use cases (Borg et al. 2014; Cleland-Huang et al. 2014). These approaches mostly focus on the optimization of recall (Borg et al. 2014). Their precision is low which makes the approaches not applicable when continuously using links (Cleland-Huang et al. 2014). A vetting of the created link candidates by an expert is necessary before their usage. In safety-critical domains such as aeronautics and the automotive industry, complete link sets are required. These links are only created periodically, when needed for certification to justify the safe operation of a system (Briand et al. 2014). Therefore, the additional effort to remove many false positive links is accepted.

Nowadays, many software companies use issue tracking systems (ITS) to specify their requirements (Maalej et al. 2014). For open source projects, the usage of an ITS is crucial and de facto standard (Merten et al. 2016a). In ITS, the requirements text is unstructured and requirement issues are mixed with other issues, for e.g. bug tracking, task and test management (Merten et al. 2016a). Furthermore, for many development activities, it is helpful to consider the links between requirements and source code *during development*, e.g. in maintenance tasks, for program comprehension, and re-engineering (Mäder and Egyed 2015; Ebner and Kaindl 2002).

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

We developed such an approach based on the recording of developer interactions while working on a requirement in an integrated development environment (IDE). The recorded interactions are then assigned to the requirement worked on. The code files touched during interactions assigned to this requirement are used to create trace links. The idea is that during these interactions developers touch files relevant to the requirements they implement. In the first version, our interaction-based trace link creation approach (*IL*) relies on developers manually selecting the requirement in their IDE before they start to work on it. This approach is implemented in a corresponding tool.[1]

In an initial study (Hübner and Paech 2017), based on open source data, we could show that *IL* including recall improvement techniques created links with perfect precision and good recall (i.e. of at least above 80%), if the developers use the requirements selection systematically.

In a second study (Hübner and Paech 2018) with students as developers, the developers did not perform the manual selection of the requirements systematically. This led to the creation of wrong trace links by our *IL* approach (precision of 43.0%, recall of 73.7%). With the integration of wrong link detection techniques into *IL*, we could improve precision to 68.2%, but this reduced recall to 46.5% (Hübner and Paech 2018). Therefore, we looked for a way to remove the error-prone manual selection of the requirements by the developers.

It is a common developers' convention to use issue identifiers (IDs) in commit messages to link commits to requirements and bug reports (Bird et al. 2009; Merten et al. 2016a; Rath et al. 2017, 2018). Trace links then can be created automatically by linking all files affected

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

by a commit with the requirement specified by the issue ID in the commit message. This does not require the awareness of the developers for interaction recording and no further effort than providing the issue IDs. However, as confirmed by recent research, typically only 60% of the commits are linked (Rath et al. 2018). Therefore, our idea was to combine the ID-based linking with interaction recording in the second version of our approach called IL_{Com}.

IL_{Com} uses the issue IDs from the developers' commit messages instead of manually selected requirements. All code files touched in the interactions before the commit are associated to the requirement identified through the issue ID.

In this paper, we report on the evaluation of IL_{Com} in a new study with students. In this new study, IL_{Com} with the improvement techniques finally achieved a precision of 90.0% and a recall of 79.0%.

We also compared our approach with pure commit-based linking and IR. Commit-based linking achieved a precision of 67.5% and a recall of 44.3% and IR a precision of 36.9% and a recall of 55.7%. Thus, in our new study we show that IL_{Com} achieves very good precision and recall and is much better in both precision and recall than the standard techniques.

In addition to the new study, we discuss the maintenance of trace links which is needed for continuous link creation and usage. Links have to be maintained along with the changing of linked artefacts during the progress of a project (Wohlrab et al. 2016; Maro et al. 2016). Trace link maintenance (TM) is a crucial part of the complete trace link management process. Without keeping trace links up to date along with changes in the linked artefacts, trace links become obsolete and useless (Cleland-Huang et al. 2014). A simple approach to link maintenance would be to remove all existing links and to regenerate the links as in the initial generation. However, this simple link maintenance approach is often not practical due to the required resources for a complete regeneration of links.

For standard link approaches, these resources are the required computing power, and the time spent and knowledge required for vetting links manually. For continuous creation and usage, the complete regeneration of links is even more impractical. Therefore, it is important to enhance trace link approaches with maintenance.

To identify suitable TM approaches, we conducted a systematic literature review (SLR). In the SLR, we answer the general research question: Which approaches do exist for trace link maintenance and what are their characteristics? Overall we identified 16 distinct TM approaches and assessed them with regard to their integration with IL_{Com}. In this paper, we report on their assessment and sketch a TM process for IL_{Com} which also uses our improvement techniques. With a future implementation of this TM process, our approach will have integrated trace link maintenance and very likely better precision and recall than without maintenance support and will therefore be suitable for continuous traceability.

The part of the paper on the evaluation of IL_{Com} has already been published in Hübner and Paech (2019). This publication is extended by the SLR and a discussion on trace link maintenance and the integration of maintenance capabilities in IL_{Com}.

The remainder of the paper is structured as follows. Section 2 introduces the basics for the evaluation of automatically created trace links and reports on related work in the context of interaction data usage and commit-based link creation. Section 3 presents the TM SLR. Section 4 gives an overview of the general *IL* approach and illustrates the details of IL_{Com}, its implementation and discusses the integration of TM capabilities found in the SLR into IL_{Com}. Section 5 outlines the experimental design of our new evaluation study for IL_{Com}. Section 6 presents and discusses the study results including threats to validity. Finally, Section 7 concludes the paper and gives an outlook on our further research.

2 Background

In this section, we introduce the basics of trace link evaluation and report on related work in the context of interaction data usage and commit-based trace link creation.

2.1 Trace Link Evaluation

In the following, we sketch the basics of trace link evaluation as already described in our paper (Hübner and Paech 2018). To evaluate approaches for trace link creation (Borg et al. 2014; Cleland-Huang et al. 2014), a gold standard which consists of the set of all correct trace links for a given set of artefacts is important. To create such a gold standard, it is necessary to manually check whether trace links exist for each pair of artefacts. Based on this gold standard, precision and recall can be computed.

The set of links found by an approach can be divided as follows: True positives (TP) are the correct links, false positives (FP) are the incorrect links. Furthermore, false negatives (FN) are correct links which were not found. Then precision (P) and recall (R) are defined by the following equations. Precision is the fraction of found links which are correct and recall is the fraction of correct links which have been found.

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

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 to consider R. Therefore, we choose $F_{0.5}$ which weights P twice as much as R. In addition, we also calculate F_1-scores to compare our results with others. For approaches using structured (Huffman Hayes et al. 2006) and unstructured (Merten et al. 2016b) data for trace link creation, good R values are between 70 and 79% and good P values are between 30 and 49%.

2.2 Related Work

In our previous papers (Hübner and Paech 2017, 2018), we already discussed related work on *IR* and the usage of interactions. For *IR* trace link creation, the systematic literature review of Borg et al. (2014) gives an overview of *IR* usage and results. Several approaches concern developer interaction. In an observation study, Soh et al. (2017) showed that observed interaction durations do not always correspond to recorded interaction durations. Konôpka and Bieliková (2015) use interaction logs to detect relations between code files and software modules, not explicitly defined in the code. Omoronyia et al. (2009) published an approach in which interactions are used to visualize and navigate trace links. In a follow up paper (Omoronyia et al. 2011) of the same authors, they also use interactions for trace link creation. They consider developer collaboration and rank interaction events. Their approach achieves a precision of 77% in the best case which is still not as good as our results for ILC_{om}.

Commit-based linkage of requirement and source code is particularly relevant for ILC_{om}. Rath et al. (2017) have studied commit linkage. They report on a data set *Ilm7* they created from seven open source projects for the purpose of evaluating traceability research. They used the issue IDs in commit messages to link issues to code files. They report that only 60% of the commits contain an issue ID.

In their follow-up work (Rath et al. 2018), they use the *Ilm7* data set to train different machine learning classifiers to countervail the problem of commits without issue IDs. To train their classifiers, they not only used the files and issue IDs from commits, but also the textual similarity (*IR*) between different artefacts (i.e. the commit message text, the issue text, the source code text) and further data, like developer-specific information. In their final experiment, they used the trained machine learning classifiers to identify the matching issues for commits without issues and achieved an averaged recall of 91.6% and a precision of 17.3% in six runs with different data sets. This precision is by far not sufficient for our purposes. A direct comparison with *IR*-based link creation is missing. However, since these results are quite similar to what others have achieved when relying on *IR* (Merten et al. 2016b) and ITS data only, it seems that the usage of *IR* to train machine learning classifiers results in the same low precision values as when relying on *IR* only.

3 Trace Link Maintenance

This section provides a systematic literature review (SLR) on the maintenance of trace links (TM). Links have to be maintained along with the changes of linked artefacts during the progress of a project (Wohlrab et al. 2016; Maro et al. 2016). They can become obsolete due to changes in the linked artefacts and thus become useless. TM is a crucial part of the complete trace link management process (Cleland-Huang et al. 2014). Therefore, this SLR collects the state of the art of TM.

Another reason to create this overview is to identify approaches suitable for the maintenance extension of our ILC_{om} approach. TM approaches are suitable for the integration with ILC_{om} if they maintain links between the same artefacts, use the same data sources, and do not disturb developers with additional effort to maintain links. Thus, the presented assessment of TM approaches is designed to review the approaches found for these criteria.

The following Section 3.1 describes the research method. Section 3.2 describes the process of the publication search and selection and Section 3.3 provides the results.

3.1 Method

The SLR follows the guidelines of Kitchenham and Charters (2007). Section 3.1.1 first introduces the review method we used. Section 3.1.2 states the research questions and their rationales.

3.1.1 Review Method

The SLR follows the approach of Kitchenham and Charters (2007). As shown in Fig. 1, the first step is to define research questions. The second step is the setup of a search strategy. Kitchenham and Charters (2007) suggest to use online databases to initially identify relevant publications. To query an online database, a search string consisting of search terms reflecting the research questions and the attributes required to answer the research questions

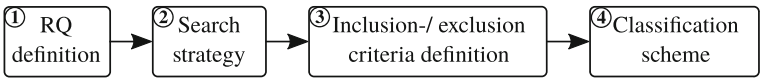

Fig. 1 SLR review method approach

is necessary. Due to the different search interfaces of the online databases (i.e. usage of logical expressions, restrictions to certain files and meta data attributes, etc.), an online database-specific adaptation of the search query and the usage of search interface-specific filtering options are necessary. Searching of online databases can be supplemented with manual target search.

Furthermore, inclusion and exclusion criteria for publications found by the keyword search have to be defined. Kitchenham and Charters (2007) define general criteria like the following: a publication has to be available in English full text, it has to be peer reviewed, and there should be empirical results in the publication.

The general criteria are then supplemented by criteria specific to the research questions and the attributes required to answer the research questions. For this SLR, the research question-specific inclusion criterion is that publications need to describe a TM approach.

Finally, a classification scheme for the identified publications is needed. This is necessary to compare the different approaches. In addition, it is also helpful to extract general publication data and meta-data, like the authors, year, title, venue.

3.1.2 Research Questions (RQs)

The general research question of this SLR is *What are the characteristics of TM approaches?* Table 1 shows a refinement of the general research question and the attributes of the approaches necessary to answer the research questions.

RQ1 asks for the characteristics of TM approaches. By *RQ1.1* (existing approaches) the primary publication for each TM approach is identified. The subsequent RQs from *RQ1.2* to

Table 1 Trace link maintenance SLR research questions and attributes

	Research question	Attributes
RQ1	What are the characteristics of TM approaches?	Standardized description of the TM approach
RQ1.1	Which trace link maintenance approaches do exist?	Publication, describing trace link maintenance approach
RQ1.2	Which artefacts are linked with each other?	Source and target artefacts of trace links
RQ1.3	Which other data sources/ artefacts are used for TM?	Used data sources/ artefacts
RQ1.4	How is maintenance performed and how are the artefacts and the trace links used for that?	Description of TM approach by a generic 4 step process
RQ1.5	To which extent and how are TM approaches automated?	Which part is automated? *(detection, execution)* Degree of automation *(automatic, semi-automatic, manual, indicated by manual/ monitoring for the detection steps and tool based/ manual in the determination & execution step)*
RQ1.6	How are the approaches evaluated and what are the evaluation results?	Type of evaluation *(qualitative, quantitative)* Evaluation results *(precision, recall, and f-measures if a quantitative evaluation has been performed)*

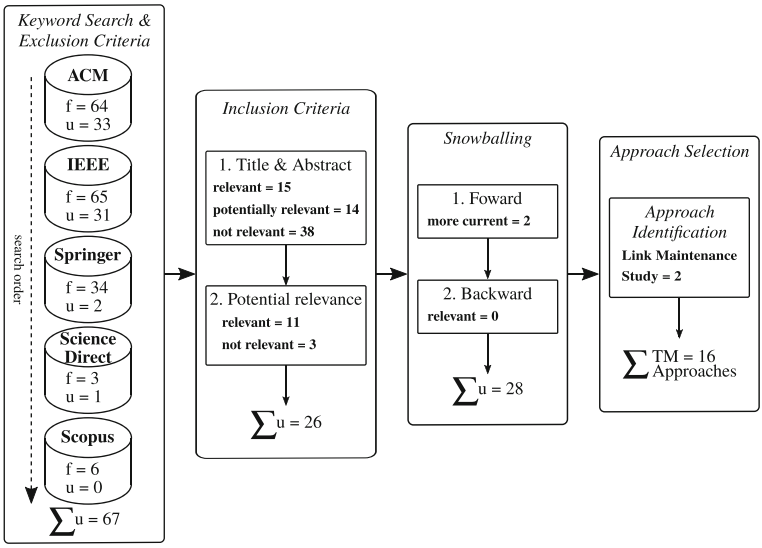

Fig. 2 Publication Search and TM Approach Identification

RQ1.5 collect the data required for a standardized TM approach description. *RQ1.2* (linked artefacts) helps to distinguish the range of use for a TM approach. With the answers from *RQ1.3* (data) and *RQ1.4* (techniques, algorithms etc.), it is possible to determine what is necessary to apply a TM approach. *RQ1.5* helps to understand which parts of a TM approach are automated. This is also interesting regarding the integration of link maintenance in our IL_{Com} approach, since one of our major goals for IL_{Com} is to add as little additional manual effort as possible. *RQ1.6* finally helps to rate the maturity of a TM approach which is also important when considering the integration into IL_{Com}.

3.2 Publication Search

Figure 2 shows the overview of the publication search and the selection of TM approaches from the identified publications. The steps of the publication search and the identification of TM approaches are described in the following.

Listing 1 shows the key word query used for the publication search. The query was tested with a pre-search by using Google Scholar.[2] Due to the amount of publications found in the pre-search, the search terms have been kept.

As shown in Fig. 2, five scientific online databases have been used to search for publications with the keywords from above in the shown order. For some of the used databases, the key word query of Listing 1 was adapted due to the database's search engines technical concerns, e.g. for some of the databases it was necessary to explicitly select the attributes of publications (like title, abstract, content, etc.) to be used when searching. The numbers for each database refer to the publications found(f) and used(u). If publications were found in multiple databases, only their first occurrence is counted. This resulted in 67 distinct publications received from all scientific online databases.

[2] https://scholar.google.com/

```
"traceability maintenance" OR "trace link maintenance"
```

Listing 1 Keyword Search Query

To filter the identified publications, the exclusion and inclusion criteria shown in Table 2 have been applied. To ensure quality and timeliness, the exclusion criteria E_1 till E_3 have been applied within the filtering functionality of the databases' search interfaces. Thus, the number of publications used shown in the first step of Fig. 2 already comprise the application of E_1 till E_3.

The inclusion criterion I_1 was applied in a second step after the overall number of 67 publications was received. First, this criterion was only applied to the publication's title and abstract. 15 publications were identified as relevant, 14 publications as potentially relevant and 38 as not relevant. After that, for the 14 potentially relevant publications the complete publication text was used to decide on their inclusion. This resulted in further 11 relevant and 3 not relevant publications. Thus, finally 26 relevant publications were identified after the application of the exclusion criterion.

To ensure that we got all relevant publications and the latest publication for a TM approach, we performed forward snowballing and backward snowballing. By forward snowballing, we identified two more current publications for two of the identified approaches, but no publication on approaches that had not yet been covered. Backward snowballing did not bring any new insights. This finally resulted in 28 relevant publications after snowballing.

The 28 relevant publications contained 16 distinct TM approaches whereof 4 of the 16 approaches were described in multiple publications. Two publications described studies about TM, but did not contain an own TM approach. For the 16 TM approaches, one primary publication was chosen based on the novelty of the publication, the comprehensiveness of the TM approach description, and the performed evaluation.

3.3 Results

This section presents the results of the SLR. First, we introduce a generic TM process which is used in Section 3.3.1 to answer the research questions and in Section 3.3.2 to discuss the results. The process consists of 4 steps separated into an impact detection and an execution part summarized in the following:

Impact Detection consists of the *detection of a change* in linked artefacts and the subsequent *detection of impacted links*. The resulting output contains the impacted artefacts and links and potential further data.

Table 2 Exclusion (E_n) and inclusion (I_n) criteria for trace link maintenance publications

Criteria	Description
E_1	publication is published before 2000
E_2	publication is not written in English
E_3	publication is not peer reviewed
I_1	publication describes a trace link maintenance approach or a trace link creation approach which includes trace link maintenance

Change Execution consists of the *determination of necessary link changes* based on the previously generated output and the *execution of those changes*.

We assume that trace links are maintained along with the maintenance of all other artefacts in the progress of a project. Thus, an artefact change triggers the maintenance of trace links. However, there are TM approaches which only analyse the current state of the project artefacts and then rate existing and find missing links. In these cases, the authors do not propose the application of their approaches for every project change. We treat them as TM approaches by adding a manual change detection step.

3.3.1 TM Approach Overview and Answers to the Research Questions

RQ1.1: Which TM Approaches do Exist Table 3 answers *RQ1.1* (which TM approaches do exist) by showing the overview of the 16 identified TM approaches and the primary

Table 3 Primary publications for identified TM approaches

ID	TM approach primary publication
P01	O. Armbrust, A. Ocampo, J. Munch, M. Katahira, Yumi Koishi, and Y. Miyamoto. Establishing and maintaining traceability between large aerospace process standards. In *ICSE Workshop Traceability in Emerging Forms of Software Engineering*, pages 36–40, 2009
P02	D. Blouin, M. Barkowski, M. Schneider, H. Giese, J. Dyck, E. Borde, D. Tamzalit, and J. Noppen. A Semi-Automated Approach for the Co-Refinement of Requirements and Architecture Models. In *25th IEEE International Requirements Engineering Conference Workshops (REW)*, pages 36–45. IEEE, 2017
P03	J. Cleland-Huang, C. K. Chang, and M. Christensen. Event-based traceability for managing evolutionary change. *IEEE Transactions on Software Engineering*, 29(9):796–810, 2003
P04	J. Cleland-Huang, P. Mäder, M. Mirakhorli, and S. Amornborvornwong. Breaking the big-bang practice of traceability: Pushing timely trace recommendations to project stakeholders. In *20th IEEE International Requirements Engineering Conference (RE)*, pages 231–240. IEEE, 2012
P05	N. Drivalos-Matragkas, D. S. Kolovos, R. F. Paige, and K. J. Fernandes. A State-based Approach to Traceability Maintenance. In *6th ECMFA Traceability Workshop (ECMFA-TW)*, pages 23–30. ACM, 2010
P06	M. Fockel, J. Holtmann, and J. Meyer. Semi-automatic establishment and maintenance of valid traceability in automotive development processes. In *2nd International Workshop Software Engineering for Embedded Systems (SEES)*, pages 37–43, 2012
P07	V. Gervasi and D. Zowghi. Supporting traceability through affinity mining. In *22nd IEEE International Requirements Engineering Conference (RE)*, pages 143–152. IEEE, 2014
P08	A. Ghabi and A. Egyed. Code patterns for automatically validating requirements-to-code traces. In *27th IEEE/ACM International Conference on Automated Software Engineering*, pages 200–209, 2012
P09	M. Kleffmann, M. Book, and V. Gruhn. Towards recovering and maintaining trace links for model sketches across interactive displays. In *7th International Workshop Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 23–29, 2013
P10	P. Mäder and O. Gotel. Towards Automated Traceability Maintenance. *Journal of Systems & Software*, 85 (10):2205–2227, 2012
P11	S. Maro and J. Steghöfer. Capra: A Configurable and Extendable Traceability Management Tool. In *24th IEEE International Requirements Engineering Conference (RE)*, pages 407–408. IEEE, 2016
P12	R. F. Paige, N Drivalos, D. S. Kolovos, K. J. Fernandes, C. Power, G K. Olsen, and S. Zschaler. Rigorous identification and encoding of trace-links in model-driven engineering. *Software & Systems Modeling*, 10(4): 469, 2011
P13	M. Rahimi, W. Goss, and J. Cleland-Huang. Evolving Requirements-to-Code Trace Links across Versions of a Software System. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 99–109, 2016
P14	H. Schwarz, J. Ebert, and A. Winter. Graph-based traceability: a comprehensive approach. *Software & Systems Modeling*, 9(4):473–492, 2009
P15	A. Seibel, R. Hebig, and H. Giese. Traceability in Model-Driven Engineering: Efficient and Scalable Traceability Maintenance. In *Software & Systems Traceability*, pages 215–240. Springer, 2012
P16	P. Ying, T. Yong, and Y. Xiaoping. Software Artifacts Management Based on Dataspace. In *WASE International Conference on Information Engineering (ICIE)*, volume 2, pages 214–217. IEEE, 2009

Table 4 Trace link source and target artefacts

Linked Artefacts	P01	P02	P03	P04	P05	P06	P07	P08	P09	P10	P11	P12	P13	P14	P15	P16	∑
Software artefacts					✓									✓		✓	3
Development standard	✓																1
Source code			✓					✓			✓		✓			✓	5
Requirements		✓	✓	✓		✓	✓	✓		✓			✓				8
(UML) Model element		✓		✓	✓	✓				✓	✓	✓		✓	✓		9
Sketch									✓								1

publication for each approach. In the following, each of the 16 TM approaches is summarized based on the characteristics shown in the Tables 4 (linked artefacts) and 5 (TM process).

In Armbrust et al. (2009) describe a manual trace link maintenance approach. Authors performing changes in specification document sections additionally have to assign link flags in all linked document sections indicating the type of change in the original sections. Link flags in a section then can be processed by another author (i.e. an expert decides about the required link change).

In Blouin et al. (2017) describe an approach in which affected link impact detection for linked requirements and architectural models is automated. Impact detection rules are based on the linked model elements' hierarchy and model element types to detect architectural refinements. Impact detection rules are triggered by monitoring changes on linked artefacts. In the execution of link changes, architectural refinement-specific rules are used to automatically maintain links.

Table 5 Trace link maintenance process

		Action	P01	P02	P03	P04	P05	P06	P07	P08	P09	P10	P11	P12	P13	P14	P15	P16	∑
S1		Manual Indication of changed artefacts	✓							✓				✓	✓		✓		5
		Monitoring of changes in artefacts		✓	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓	13
S2		Manual indication of impacted links	✓																1
		Monitoring of impacted links		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	15
		Impact detection rules		✓		✓	✓	✓	✓	✓		✓		✓	✓	✓	✓	✓	12
	Data	Model element type		✓		✓	✓	✓				✓		✓		✓	✓		8
	Data	Model element hierarchy (parent/child)		✓													✓		2
	Data	Term pairs from linked artefacts							✓										1
	Data	Source code structure								✓					✓				2
	Data	Interaction type & sequences										✓							1
	Data	2 artefact versions													✓				1
	Data	Software artefact type													✓			✓	2
O		Change type (for affected links, flag/score)	✓	✓			✓	✓	✓	✓		✓		✓	✓	✓	✓	✓	12
S3		Tool based execution of (change type specific) rules		✓			✓		✓	✓		✓			✓	✓	✓	✓	9
	Data	Model element type					✓					✓				✓	✓		4
	Data	Model element hierarchy (parent/child)		✓													✓		2
	Data	Linked artefacts score threshold							✓	✓									2
	Data	Software artefact type													✓			✓	2
		Manual use of output	✓		✓	✓		✓			✓		✓	✓					7
S4		Manual change of links	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓		✓		✓	12
		Tool based link change		✓			✓		✓	✓		✓			✓	✓	✓	✓	9

*S1: Detection of change; S2: Detection of impacted links; O: Output of detection (as input for execution); S3: Determination of necessary link change; S4: Execution of change

*S2 Data: for impact detection rules other than linked artefacts; S3 Data: other than linked artefacts, impact detection data & change type

In Cleland-Huang et al. (2003) describe an approach in which links between requirements and source code are maintained manually. The only automated part is in the impact detection part of the approach: by monitoring that whenever a linked artefact is changed a notification is generated and sent to the original author of the artefact. Based on the notification, the original author then has to maintain the involved links manually.

In Cleland-Huang et al. (2012), the same authors extend the approach Cleland-Huang et al. (2003), The only difference is that the notification in this approach contains hints on how to perform the necessary link change (change type).

In Drivalos-Matragkas et al. (2010) describe an approach in which links between different kinds of model elements are maintained automatically using model element type-specific impact detection rules. Impact detection rules triggered by monitoring changes on linked artefacts can detect certain link change types. For a set of defined change types, further rules exist which are used to then automatically execute the link change. For the other change types, a notification is generated and the change has to be performed manually.

In Fockel et al. (2012) describe an approach in which links between different kinds of model elements and textual requirements are maintained semi-automatically. To detect impacted links, this approach uses rules utilizing the model element type and outputs a change type passed on to the then manually performed link change. The constraint validation rules are triggered by monitoring changes on linked artefacts.

In Gervasi and Zowghi (2014) describe an approach in which links between textual requirements are maintained automatically. For impact link detection, the approach uses rules that utilize term pairs in linked artefacts. The approach is triggered by monitoring changes on linked artefacts. Link impact detection outputs a score for artefact pairs that is based on the linked term pairs. For link change execution, rules are used along with the score to automatically remove and add links.

In Ghabi and Egyed (2012) describe an approach in which links between source code and requirements are maintained automatically. The approach is triggered manually. For impact detection, this approach uses source code structure-based rules along with existing links to calculate a numerical value indicating a change type. For link change execution, a threshold and the previously calculated numerical value are used to automatically remove and add links.

In Kleffmann et al. (2013) describe an approach in which links between sketches (of software systems) on interactive displays are maintained manually. The approach supports link maintenance in the impact detection part: whenever a user changes a linked sketch, all affected links and linked other sketches are highlighted. Potential change to links is then performed manually by the user.

In Mäder and Gotel (2012) describe an approach in which links between UML model elements and textual requirements are maintained semi-automatically. Rules based on interaction sequences are used to detect certain change types. The change type detection rules are triggered by monitoring changes on linked artefacts. Change type-specific rules are used to automatically perform link changes. If there are no link change rules for a detected change type, a user notification is generated and the link change is performed manually.

In Maro and Steghöfer (2016) describe an approach in which links between different model elements defined in an extendable meta-model are maintained manually. Similar to Cleland-Huang et al. (2003) the only automated part is the initial impact detection. Whenever a linked artefact is changed, a notification is generated. The execution of the link change then has to be performed manually by a user.

In Paige et al. (2011) describe an approach in which links between model elements defined in different meta-models are maintained semi-automatically. Similar to Drivalos-Matragkas et al. (2010) model element type-specific impacted detection rules are used. Impact detection rules are triggered by monitoring changes on linked artefacts. The output of the link impact detection is a model element-specific change type. This change type is used to manually change the links of the respective model element.

In Rahimi et al. (2016) describe an approach in which links between source code and requirements are maintained automatically. The approach uses rules, two versions of the source code or requirements, source code structure, and the artefact type to detect refactorings. Refactoring-specific rules are used to automatically change links. The rules for refactoring detection can be triggered manually or by monitoring changes on linked artefacts.

In Schwarz et al. (2009) describe an approach in which links between different model elements are maintained semi-automatically. All model elements are represented as a graph in the approach. For impacted link detection, model element type-specific rules are used. The rules for impact detection are triggered by monitoring changes in target models, followed by subsequent changes in source models on the same model element. Link changes originating in source models are performed automatically by rules, link changes originating in target models have to be performed manually.

In Seibel et al. (2012) describe an approach in which links between different model elements are maintained automatically. Impacted link detection rules use hierarchy relations between different kinds of model elements to determine affected linked model elements. For each linked model element, a change type is generated. The change type is used together with model element type-specific rules and again hierarchy information of model elements to automatically maintain links for each model element. The rules for impacted link detection can be triggered manually or by monitoring changes on linked artefacts.

In Ying et al. (2009) describe an approach in which links between source code and other software artefacts defined in an extendable ontology are maintained semi-automatically. Rules utilizing artefact type data extracted from an ontology are used to create change types for existing links by monitoring changes on linked artefacts. A further set of artefact type-specific rules is used to perform the link change automatically. The approach includes a set of predefined artefact type-specific link change rules. If for an artefact type no link change rules are defined, a user notification is generated and the link change has to be performed manually.

RQ1.2: Linked Artefacts Table 4 shows the linked artefacts of the TM approaches. These artefacts can be software artefacts in general, development standards, source code, requirements specifications in a textual format, (UML) model elements, and sketches which are hand drawn graphical and textual descriptions of software system parts.

The most frequently used artefacts in the approaches are model elements (9) and requirements (8). Almost all approaches link at least 2 different artefacts. The combinations of model element and requirement (4) and requirement and source code (3) are the two most common ones.

RQ1.3 Other Data Sources Used, RQ1.4 Performance of TM and RQ1.5 Approach Automation Table 5 shows the TM process characteristics for the approaches as introduced in Section 3.3.1. The table distinguishes the performed actions within the steps along with the data used. The number of actions or data shown in the right sum column do not always sum up to 16. The reason is that some approaches support multiple actions in the steps,

e.g. in approach Rahimi et al. (2016), step *S1* can be triggered manually and by monitoring changes in artefacts as well. For data used in step *S2* and *S3*, only some of the approaches use data in addition to the linked artefacts.

In the following, we summarize the characteristics of the 16 TM approaches within the four generic TM process steps. This comprises the answers to research questions *RQ1.3* on the artefacts used, to *RQ1.4* on how trace link maintenance is performed and to *RQ1.5* on the automation degree of the TM approaches.

Step 1: Detection of Change 13 of 16 approaches monitor the changes on linked artefacts. Two of these 13 approaches can be triggered manually as well. For the other three approaches, only a manual indication of changes on linked artefacts is performed.

Step 2: Detection of Impacted Links 15 approaches automatically detect the links involved. In one approach, this detection is performed manually. 12 of these 15 approaches use rules for the impact detection. Regarding the data sources, most (8) of the impact detection rules are based on model element type-specific linkage, two approaches additionally use artefact hierarchy and one of these further uses the interaction type. Other approaches use term pairs from already linked artefacts, two versions of artefacts, and the software artefact type. For the other 3 approaches without impact detection rules, a notification with the impacted artefacts and links is generated and sent to the user to trigger link change determination.

Data: Output of Detection (as input for execution) All approaches output the affected artefacts and links. In addition, 12 of the 16 approaches assign a change type to the affected links. This change type can indicate how the link should be maintained, e.g. *delete link* or it can indicate what kind of check to perform on a linked artefact, e.g. *check for changes on all description texts of linked artefacts*, or it can be a numerical value which later can be used to perform the link maintenance. The four approaches without a change type generate a user notification with the impacted artefacts and links and link maintenance is performed manually by a user. Three of these four approaches are without impact detection rules, one has impact detection rules and generates additional instructions on how a user should maintain the links manually in its notification.

Step: 3: Determination of Necessary Link Change Nine of the 16 approaches determine the necessary link change automatically by a tool-based rule execution. In seven approaches, the detection output is used manually.

From the nine approaches with rules, most (4) use model element type-specific linkage for link change determination, one of them uses hierarchy information of different artefacts in addition. Other approaches use the software artefact type (2) or a threshold to judge about an affinity score for links calculated in the impact detection step.

Step 4: Execution of Change Four of the 16 approaches perform the link change completely automated without any required user interaction by using detected change types for links. Five of the 16 approaches perform the link change only in cases where a known change type has been detected. They inform the user about cases which cannot be handled automatically by showing the affected artefacts and links. In the other seven approaches, the execution of link change is performed manually by the user. In two of these seven approaches a change type is included in the user notification, but has to be processed manually.

Regarding the techniques for impact detection and change execution rules, the approach descriptions in the publications often are on a conceptual level. We summarize the

mentioned techniques in the following. For impact detection based on model element type and rules, constraint checks are used. Pattern recognition is used with source code structure. Approaches based on the textual contents of artefacts use IR and natural language processing to determine textual similarity. Some of the model-based approaches store the impact detection rules in specific meta-model element attributes. Other model-based approaches transfer the model elements to a graph representation and then use graph merging and other graph-based techniques in their rules. One approach uses an ontology and reasoning to manage different software artefact types and rules for the implementation of impact detection.

RQ1.6 Performed Evaluations Table 6 shows whether an evaluation has been performed for an approach and the properties of this evaluation. Basic properties are the research process (qualitative/quantitative) and the data collection method used. For all approaches which were not evaluated except Ying et al. (2009), the authors provided a running example.

For nine of the 16 approaches, an evaluation was performed. The evaluations were purely quantitative (4), purely qualitative (3), and mixed (2). The qualitative evaluations always consider the approaches' feasibility. The quantitative evaluations concerned mostly precision and recall and in one case only scalability and performance. Data was often collected in an experiment with developers (3) or a simulation of the approach (4). Twice an interview was used and only once was the approach used in a real world application of the approach with developers. For simulation and real world application, precision and recall was determined.

3.3.2 Discussion

Figure 3 summarizes the key facts of all 16 evaluated TM approaches in our generic TM process. Monitoring of changes and monitoring of impacted links is quite prevalent. However, the data used and the techniques differ. Also, the output of a change type is very common. Tool-based change determination is still prevalent, but less common than monitoring. Again, different data is used. Interestingly, only few approaches perform changes fully automated. In most approaches at least for some cases, manual change is necessary. This might also be the reason why only one approach was applied in industry. The approaches are quite different. Our generic TM process is the first to provide a unifying view. It can be used to integrate the ideas of different approaches. In the next section, we discuss such an integration for our ILC_{om} approach.

Table 6 Performed evaluations

Evaluation Properties		P01	P02	P03	P04	P05	P06	P07	P08	P09	P10	P11	P12	P13	P14	P15	P16	∑
Research Process	Qualitative	✓	✓				✓				✓					✓		5
	– Feasibility	✓	✓				✓				✓					✓		2
	Quantitative[1]				✓			✓	✓		✓			✓		✓		6
	– Scalability/ Performance															✓		1
	– Precision				72			85	95		96			86				5
	– Recall				22			64	96		79			88				5
Data Collection	Experiment with Developers	✓									✓					✓		3
	Application with Developers										✓							1
	Questionnaire/ Interview	✓					✓											2
	Simulation of the approach				✓			✓	✓					✓				4

[1]For precision and recall, the best values achieved are shown, in some cases these values were achieved in different runs with different settings

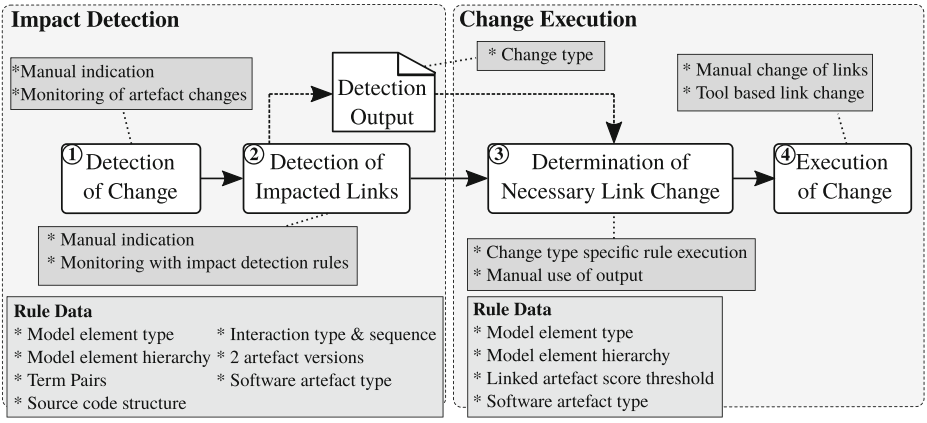

Fig. 3 Trace Link Maintenance Process with TM Approach Data

3.4 Threats to Validity

Even though a primary concern of conducting an SLR is to create reliable and reproducible results when answering research questions, there are possible threats to the SLR's validity.

First, regarding the performed search, the choice of suitable search terms and search sources can bias the results. That is to say, publications may not be found due to term mismatches or because they are only listed in a source not used when searching. However, we also used synonyms like *evolution* instead of *maintenance* to check the suitability of our chosen search terms and *Google Scholar*[3] to avoid missing relevant publications completely.

Second, the evaluation of TM approaches can be flawed, e.g. by misunderstanding the description of an approach. We therefore created explicit TM approach descriptions in an iterative way and involved two researchers in the process. Initially, one researcher created a comprehensive textual description of the approaches and ensured that the descriptions were plausible. These were reviewed by the second researcher. Then, we used these descriptions to create our generic TM process and characterized each TM approach according to this process. Finally, the first researcher used the TM process-based approach characterizations to create a short standardized textual description for each approach which was again reviewed by the second researcher.

4 Interaction Recording-based Trace Link Creation and Maintenance

The following Section 4.1 presents an overview of the *IL* approach and Section 4.2 illustrates the details of IL_{Com}. Section 4.3 discusses the integration of maintenance capabilities found in the previously presented SLR into IL_{Com}.

4.1 IL-approach Overview

Figure 4 shows the overview of our *IL* approach consisting of three steps. In the first step *Interaction Capturing*, interaction events in the IDE of a developer are captured and associated to a requirement.

[3] https://scholar.google.com/

In the second step *Trace Link Creation*, all interaction events captured for a requirement are used to generate trace links between the requirement and the source code files affected by the interactions. Before the trace link generation, interactions with files not directly created by the developers or files such as build configurations, project descriptions, readme files, meta-data descriptions, binaries etc., which are not source code files and files from 3rd parties, such as libraries, are removed. Then, the recorded interactions for a file are aggregated into a single link. In the aggregation, interaction event-specific meta data such as the event type (edit or select), the duration as the sum of all events' durations based on the interactions' time stamps for specific files, and the frequency of how often an interaction occurred for a specific file are captured for the resulting links. The result of this second step is a list of trace links (including the meta data aggregated from the interactions) which are used as input for the third step *Trace Link Improvement*.

In the third step, precision is improved by removing potentially wrong links using the interaction-specific meta data frequency, duration, and event type from the links created in the previous step. Precision is also improved by using the source code structure, i.e. the references from one source code file to other source code files. Finally, the source code structure is used to improve the recall. Details about the improvement techniques and their impact on precision and recall can be found in our previous study (Hübner and Paech 2017) for recall improvement and Hübner and Paech (2018) for precision and recall improvement. We explain the most relevant techniques together with the results in Section 6.1.

The first implementation of the *IL*-approach relied on the developers' manual indication of the requirement they work on during their interactions. This was implemented as plug-in for the IntelliJ IDE. We extended an existing activity tracker plug-in to also track the interactions with requirements. In addition, we used the *Task & Context* functionality of IntelliJ to associate interactions with requirements. The developers could connect IntelliJ to the Jira project and the developers had to select the specific Jira issue with the UI of the *Task & Context* functionality when working on a requirement. As a result, the interaction log contained activation and deactivation events for requirement issues. These activation and deactivation events were used to allocate all interactions between the activation and deactivation events for a specific requirement to this specific requirement. Since the developers also used sub-task issues and since sub-tasks describe details for implementing the requirement, we combined the interactions recorded for requirements and for the corresponding sub-tasks.

This first version of our approach with manual interaction assignment resulted in unsatisfying results for precision (Hübner and Paech 2018). As we assumed that the main reason is the effort necessary for the manual assignment, we developed the commit-based interaction assignment version IL_{Com}. The next section reports about IL_{Com}'s details.

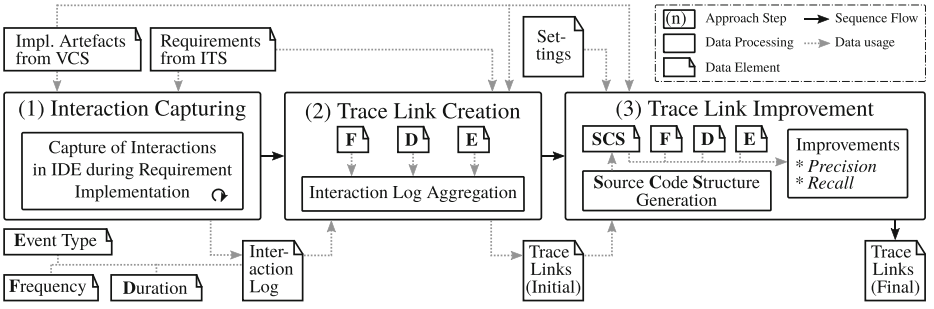

Fig. 4 *IL* Approach Overview: Interaction Capturing, Trace Link Creation and Improvement

4.2 Commit-based Interaction to Requirement Assignment with IL_{Com}

In this section, we introduce the commit-based variant of our approach, called IL_{Com}. The difference between IL and IL_{Com} lies in the first interaction capturing step. IL_{Com} uses recorded interactions and issue IDs in commit messages for link creation. In IL_{Com}, interactions are recorded until a developer performs a commit. If the commit message contains an issue ID, all recorded interactions are associated to this issue ID and the history of recorded interactions is cleared. If multiple issue IDs are contained in the commit message, the recorded interactions are associated to all issue IDs. If no issue ID is contained in the commit message, interaction recording continues until there is a commit with a commit message containing an issue ID. Clearly, this can impact precision and recall, as the commits without ID might be associated with another issue (Herzig and Zeller 2013; Kirinuki et al. 2014). This will be discussed in Section 6.2. After the association of issue IDs with interactions, link creation can be performed as described for IL in Section 4.1.

We implemented the interaction capturing for IL_{Com} in the current study as plug-in for the Eclipse IDE. Our tool bundles all recorded interactions and uploads them to the Jira issue specified by the Jira issue ID in the commit message. The interaction events recorded by our tool comprise a time stamp, the type of interaction (select or edit), the part of the IDE in which the interaction occurred (e.g. editor, navigator, etc.), the file involved in the interaction, and a degree of interest (DOI) metric for the file. The DOI is a numerical value calculated for a file considering the number of interactions (frequency) and the type of interactions with the file, i.e. edit interactions are rated higher than select interactions (Kersten and Murphy 2006).

4.3 Integration of Trace Link Maintenance

This section uses the results of the SLR from the previous section to discuss the integration of maintenance capabilities into IL_{Com}. In Section 4.3.1 the selection criteria for suitable TM approaches are introduced and applied to select two TM approaches. In Section 4.3.2, the integration of the selected maintenance capabilities into IL_{Com} is discussed.

4.3.1 Approach Selection

Table 7 shows the overview of the selection criteria, their description, and the rating of TM approaches for the criteria.

The first selection criterion (C1) for the transferability of a TM approach to IL_{Com} is the match of the source and target artefacts used for links. The second selection criterion

Table 7 Selection criteria and TM approach rating for the integration of maintenance capabilities in IL_{Com}

	Criteria	Description	Approach Rating		
			P03	P08	P13
C1	Linked artefatcs	Linked artefacts have to be the same as in IL_{Com} *(requirements, source code)*	✓	✓	✓
C2	Additional used data sources	Additional data sources used or techniques to maintain links have to be available in IL_{Com} (interaction logs, source code structure)	✓	✓	✓
C3	Automation	Impact on the overall automation of IL_{Com} *(Impact detection and link change should be fully automated)*		✓	✓

(C2) is the availability of the additional data sources and techniques. The third selection criterion (C3) is the degree of automation provided by a TM approach and the influence on IL_{Com}*'s* automation. These criteria enable a smooth integration of TM capabilities into IL_{Com}. Only the three TM approaches Cleland-Huang et al. (2003) and Ghabi and Egyed (2012) and Rahimi et al. (2016) shown in Table 7 satisfy the criteria C1 (match of linked artefacts). Thus, the right hand column of Table 7 shows the rating of the selection criteria for these three TM approaches only.

Of these, only the approaches Ghabi and Egyed (2012) and Rahimi et al. (2016) satisfy all selection criteria. In Cleland-Huang et al. (2003), the determination of impacted links and the execution of link changes are manual. Therefore, C3 is not satisfied. The change detection in Ghabi and Egyed (2012) is manual, but the other steps are automated. As the change detection can easily be automated in our approach, we consider C3 as satisfied. Rahimi et al. (2016) is fully automated and thus satisfies C3.

The approach Ghabi and Egyed (2012) uses source code structure in their rules for impacted link detection and thus satisfies C2. The approach Rahimi et al. (2016) requires a second version of source code files or requirements in addition to linked artefacts. Due to the usage of an ITS and a version control system along with Il_{Com}, a second version of source code files and requirements is directly available. Thus, the approach Rahimi et al. (2016) also satisfies C2. Therefore, in the following we discuss how the techniques used in Ghabi and Egyed (2012) and Rahimi et al. (2016) can be integrated into IL_{Com}.

4.3.2 Integration of Trace Link Maintenance Capabilities in IL_{Com}

In this section, we define a TM process for IL_{Com}, using the generic TM process of our SLR (cf. Section 3.3.2) and based on the data and techniques used in Ghabi and Egyed (2012) and Rahimi et al. (2016).

S1: Detection of Change After each commit, there is an automated check of the interactions whether an artefact was changed. A change is indicated by an edit action. This corresponds to the manual change detection of Ghabi and Egyed (2012) and Rahimi et al. (2016). That is to say, only approach Rahimi et al. (2016) was also intended to be used continuously during changes on linked artefacts.

S2: Detection of Impacted Links The detection of impacted links is performed by the rules from Ghabi and Egyed (2012) and Rahimi et al. (2016). As discussed for criterion C2, the data sources are available. In addition to using two artefact versions for the detection of refactorings as in Rahimi et al. (2016), in IL_{Com} it is possible to use the recorded interactions from *S1*. IDEs provide capabilities to perform certain refactorings, like *extract method to class*. Such refactorings can be detected directly in the interactions. Furthermore, one could define patterns of low level interactions which comprise the interaction sequence of a certain refactoring. These patterns could also be detected automatically in the interactions.

Data: Output of Detection The rules used in *S2* output a change type.

S3: Determination of Necessary Link Change and S4: Execution of Change The determination of link changes and the execution of the changes is fully automated for Ghabi and Egyed (2012) and Rahimi et al. (2016). The link changes derived from the affinity scores used in Ghabi and Egyed (2012) are similar to the link changes in the improvement

techniques of IL_{Com}. The IL_{Com} precision improvement *source code structure in story* removes existing links from a requirement (user story) to source code which is not connected with other source code linked to this requirements. Similarly, Ghabi and Egyed (2012) removes links if the affinity score of a method drops below a threshold because it is not connected by source code structure to other linked methods. The IL_{Com} source code structure-based recall improvement adds links by following the call relation to other source code files from source code files already linked to requirements. Similarly, Ghabi and Egyed (2012) adds links if the affinity score of a method rises above a threshold because it is connected by source code structure to other already linked methods. Since the rules executing the link change in Rahimi et al. (2016) rely on the specific refactoring created as output of the previous *S2* step, these link change rules can directly be transferred to IL_{Com} and executed whenever a refactoring is detected.

Altogether, TM can be performed fully automated in IL_{Com} using capabilities from Ghabi and Egyed (2012) and Rahimi et al. (2016). However, neither Ghabi and Egyed (2012) nor Rahimi et al. (2016) provide an implementation. Therefore, so far we have not implemented the maintenance in our approach.

5 Experimental Design

In this section, we describe the details of our new study to evaluate the trace link creation with IL_{Com}. We start with an overview of the different trace link creation techniques used in Section 5.1. This is followed by the research questions and the description of how we created the trace links in Section 5.2. In Section 5.3, the projects used are described, followed by the description of the data sources in Section 5.4 and the gold standard creation in Section 5.5. The major difference of this new study compared to our previous studies is that IL_{Com} with commit-based interaction assignment has been used instead of manual interaction assignment.

5.1 Trace Link Creation Techniques

In the following, we summarize the notations for the different link creations (*IR*, *IL*, *ComL*, and IL_{Com}) and improvement techniques (shown by subscript *i*) as used in the subsequent description of the study:

- ***IR*** denotes the approach for link creation by information retrieval and $\boldsymbol{IR_i}$ denotes that also source code structure-based improvement techniques from the third step of our approach were applied. We used vector space model (VSM) as technique for IR-based link creation in this study (Borg et al. 2014).
- ***IL*** denotes our approach for link creation with manually recorded interactions to a requirements assignment and $\boldsymbol{IL_i}$ denotes that also interaction-specific meta data and source code structure-based improvement techniques from the third step of our approach were applied.
- ***ComL*** denotes the approach for link creation by using only the issue IDs from commit messages and the files contained in the commits and $\boldsymbol{ComL_i}$ denotes that also source code structure-based improvement techniques from the third step of our approach were applied.
- $\boldsymbol{IL_{Com}}$ denotes our approach for link creation with commit-based recorded interactions to requirements assignment and $\boldsymbol{IL_{Com\,i}}$ denotes that also interaction-specific meta

data and source code structure-based improvement techniques from the third step of our approach were applied.

Note that interaction meta data-based improvement was only possible for the approaches involving interaction recording.

5.2 Research Questions

The research questions we answer in our study are:

RQ2: *What is the precision and recall of IL_{Com}- and IL_{Com_i}-created trace links?* Our hypothesis was that the initial precision of IL_{Com} improves, compared to our previous student study, since there is no additional effort for requirement selection by developers. For IL_{Com_i} compared to IL_{Com}, we expected further improvement of precision.

RQ3: *What is the precision and recall of ComL- and $ComL_i$-created trace links?* Our hypothesis was that precision and recall are lower than the precision of IL_{Com}- and IL_{Com_i}- created links respectively, as the latter uses more information (the interactions).

RQ4: *What is the precision and recall of IR- and IR_i- created trace links?* Our hypothesis was that *IR* and IR_i have a significantly lower precision and a similar recall in comparison to IL_{Com} and IL_{Com_i}.

The overall goal of this new study is to evaluate, whether the commit-based interaction and assignment and link creation by IL_{Com} improves the precision compared to the manual interaction assignment by *IL* (*RQ2*). Moreover, we investigate whether recording and using interactions outperforms link creation, which relies on commit data only (*RQ3*). Finally, we also compare the results of IL_{Com}-created links with *IR*, since *IR* serves as a baseline for automated link creation and for the comparison with our previous studies (*RQ4*).

5.3 Evaluation Project

In the following, we describe the project used for the evaluation of the IL_{Com} approach. The project was Scrum-oriented with university students as developers and a real world customer from an external company. The project lasted from October 2017 to March 2018. The aim of the project was to develop an Android-based indoor navigation app for students in university buildings. Typical use cases for such an app are navigating to the room of a certain lecture or finding any other point of interest efficiently. The customer was a navigation software development company.

Six students participated in the project. In addition, an adviser from our chair was involved. The project was split into six sprints. In each of these sprints, one of the students acted as Scrum master and thus was responsible for all organizational concerns such as planning the development during the sprint and communicating with the customer.

For all requirement management-related activities, a Scrum Jira[4] Project was used. This included the specification of requirements in the form of user stories and the bundling of the stories in epics. An example of a user story is *Show point to point route* and the corresponding epic of this story is *routing*. To assign the implementation of user stories to developers, sub-task issues were used. A sub-task comprises partial work to implement a user story, e.g.

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

Show route info box. For the implementation, the developers used Git as version control system and the Eclipse IDE with the Android software development kit (SDK). We provided an Eclipse plug-in implementing our interaction recording tools. For the usage of Git, there was an explicit guideline to use a Jira Issue ID in any commit message to indicate the associated Jira Issue.

The customer provided a proprietary Java SDK of their own for the general use case to develop Android mobile navigation apps. The developers needed two sprints to understand the complexity of the SDK and to set up everything in a way to work efficiently on the implementation of the requirements. The programming language for the logic and data management part was Java and the UI was implemented in Android's own XML based language.

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

To perform the evaluation of IL_{Com} we created a data set using the interaction recording of our tool, the requirements managed in the Jira ITS, and the source code in the Git version control system as data sources. The details of these three data sources are described in the following.

5.4 Data Sources

In this section, the three different data sources used in IL_{Com}'s evaluation are described.

Source Code in the Git Version Control System The Git repository comprised 406 commits. 226 commits (55.67% of all commits) did contain a Jira issue ID which is a similar proportion as reported by others (Rath et al. 2017). We excluded files as described in Section 4.1 from the Git repository.

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

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

Interaction Recordings The interaction recordings for the 17 stories and 74 sub-tasks comprise 6471 interaction events separated in 205 commits. After removing the interaction events whose files were out of scope as described previously (cf. Section 4.1), 4012 interaction events were left in the interaction recordings and used for link creation.

5.5 Gold Standard Creation

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

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

6 IL_{Com} Evaluation Results

In this section, the results of our evaluations are reported, the research questions are answered, and the results are discussed.

6.1 Answer to Research Questions

Table 8 shows the results for IL_{Com} and for different settings of IL_{Com_i}. We can answer the first part of *RQ2* about precision and recall of IL_{Com} with: IL_{Com} has a precision of 84.9% and a recall of 67.3% and thus a $f_{0.5}$-measure of 0.807.

Table 8 Results for IL_{Com} and IL_{Com_i} with Different Settings

Approach	Setting[1]	Precision	Recall	$F_{0.5}$	$F_{1.0}$	#Links[2]					Src Files	
						CE	TP	FP	GS	FN	Used	GS
IL_{Com}	none	0.849	0.673	0.807	0.751	245	208	37	309	101	58	66
IL_{Com_i}	*T:e*	0.904	0.460	0.758	0.609	157	142	15	309	167	58	66
IL_{Com_i}	*T:s*	0.829	0.282	0.597	0.420	105	87	18	309	222	37	66
IL_{Com_i}	*D10*	0.885	0.521	0.776	0.656	182	161	21	309	148	52	66
IL_{Com_i}	*D60*	0.901	0.411	0.727	0.564	141	127	14	309	182	50	66
IL_{Com_i}	*F2*	0.813	0.463	0.706	0.590	176	143	33	309	166	54	66
IL_{Com_i}	*F10*	0.850	0.311	0.631	0.455	113	96	17	309	213	40	66
IL_{Com_i}	*Sis*	0.904	0.485	0.771	0.632	166	150	16	309	159	40	66
IL_{Com_i}	*T:e,s; Sis;CS*	**0.900**	0.790	**0.876**	0.841	271	244	27	309	65	62	66

[1] interaction type improvement: $T{:}e|s = Type{:}edit|select$, duration improvement: $D10|D60 = dur.>= 10|60$ *sec.*, frequency improvement: $F2|10 = freq.>= 2|10$, source code structure improvements: *Sis* = *Source code structure in story*, *CS* = *Source code structure recall improvement*, the last row of the table shows the best setting for IL_{Com_i} in which precision was improved by interaction type edit and select (*T:e*, *s*) and by limiting links to source code structure in story (*Sis*), then recall was improved also by using the source code structure (CS), this setting was used for all subsequent IL_{Com_i} results

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

Similar to our previous study (Hübner and Paech 2018), we evaluated different settings for our improvement techniques as shown in the second column of Table 8. For precision improvement, links are removed using interaction-specific meta data of links. If a specific interaction only has a short duration (D10, D60), or only a low frequency (F2, F10), or is of a specific type (select T:s, edit T:e), it is less likely to correspond to a correct link. E.g. edit interactions are more important for links than select interactions. Furthermore, precision is also improved when using the source code structure (i.e. references between the files due to usage and calls). We remove links from a user story to a code file (Sis) (cf. Section 4.3.2), if this code file is not connected to any other code linked to the user story. Since a file which contributes to a requirement very likely is connected to other code files. Recall improvement also uses the source code structure and adds new links by following source code references from already linked source code files up to a certain level (CS).

Initially, we investigated the different wrong link detection techniques in isolation and then combined different techniques to achieve the overall best precision improvement. On this best precision result, we also applied our source code structure-based recall improvement by following source references until level 4. The last row of Table 8 shows this best case of IL_{Com_i}.

We can answer the second part of *RQ2* about precision and recall of IL_{Com_i} with: In the best case, IL_{Com_i} has a precision of 90.0% and a recall of 79.0% and thus a $f_{0.5}$-measure of 0.876. Thus, IL_{Com_i} improves precision by 5.1%, recall by 22.7% and $f_{0.5}$-measure by 0.069 compared to IL_{Com}.

Thus, both, our hypothesis that IL_{Com} has better quality than *IL* since there is no additional effort to assign recorded interactions and that IL_{Com_i} improves the quality of IL_{Com} further for *RQ2* are verified.

Table 9 shows the results for *ComL* and $ComL_i$ and for comparison also the previously reported results of IL_{Com}. We can answer the first part of *RQ3* about precision and recall of *ComL* with: *ComL* has a precision of 66.8% and a recall of 41.7% and thus a $f_{0.5}$-measure of 0.597. For $ComL_i$, we applied the *source code structure in story (Sis)* precision improvement followed by *source code structure* recall improvement. We can answer the second part of *RQ3* about precision and recall of $ComL_i$ with: $ComL_i$ has a precision of 67.5% and a recall of 44.3% and thus a $f_{0.5}$-measure of 0.611. The results for *ComL* also verified our hypothesis that *ComL* links have a lower quality than IL_{Com} links.

Table 10 shows the results for *IR* and IR_i and for comparison also the previously reported results of IL_{Com} and IL_{Com_i}. For IR_i we applied the same improvements as for $ComL_i$. We can answer the first part of *RQ4* about precision and recall of *IR* with: *IR* has a precision of

Table 9 Results for *ComL*, $ComL_i$ and Comparison with IL_{Com}

Approach[1]	Precision	Recall	$F_{0.5}$	$F_{1.0}$	#Links					Src files	
					CE	TP	FP	GS	FN	Used	GS
IL_{Com}	0.849	0.673	0.807	0.751	245	208	37	309	101	58	66
IL_{Com_i}	0.900	0.790	0.876	0.841	271	244	27	309	65	62	66
ComL	0.668	0.417	0.597	0.514	193	129	64	309	180	59	66
$ComL_i$	0.675	0.443	0.611	0.535	203	137	66	309	172	61	66

[1] For the application of improvement techniques, the best case is shown, for IL_{Com_i} this is the setting as introduced in the last row of Table 8, for $ComL_i$ precision was improved by limiting links to source code structure in story (*Sis*), then recall was improved also by using the source code structure (CS)

Table 10 Results for *IR*, IR_i and Comparison with IL_{Com} and IL_{Com_i}

Approach[1]	Precision	Recall	$F_{0.5}$	$F_{1.0}$	#Links					#Stories	Src Files	
					CE	TP	FP	GS	FN		Used	GS
IL_{Com}	0.849	0.673	0.807	0.751	245	208	37	309	101	17	58	66
IL_{Com_i}	**0.900**	0.790	**0.876**	0.841	271	244	27	309	65	17	62	66
IR	0.335	0.492	0.358	0.398	454	152	302	309	157	16	60	66
IR_i	0.369	0.557	0.396	0.444	466	172	294	309	137	16	64	66

[1] As *IR* technique VSM with a similarity threshold of 0.2 was used, for the application of improvement techniques, the best case is shown, for IL_{Com_i} this is the setting as introduced in the last row of Table 8, for IR_i precision was improved by limiting links to source code structure in story (*Sis*), then recall was improved also by using the source code structure (CS)

33.5% and a recall of 49.2% and thus a $f_{0.5}$-measure of 0.358. IR_i has a precision of 36.9% and a recall of 55.7% and thus a $f_{0.5}$-measure of 0.396. We can answer the second part of *RQ4* about precision and recall of IR_i with: IR_i improves precision by 3.4%, recall by 6.5% and $f_{0.5}$-measure by 0.038 compared to *IR*. In comparison to IL_{Com_i}, precision, recall, and $f_{0.5}$-measure is lower respectively. This confirms our hypothesis, that precision is lower. In addition, it shows that also recall is lower. As in our previous studies (Hübner and Paech 2017, 2018), our *IL* approaches always outperformed *IR* and IR_i. Our results achieved for *IR* are quite similar in our studies and similar to other studies as well (Merten et al. 2016b).

6.2 Discussion

Precision and recall of IL_{Com} in this study are better than *IL* in our previous study (Hübner and Paech 2018). The new study confirmed our previous results that our approach outperforms all other link creation approaches it has been compared with, i.e. *IR*- and commit-based link creation *ComL*. The fact that *IR* link creation between unstructured requirements in ITS and source code is inferior to that in structured requirement cases is reported by others (Huffman Hayes et al. 2006; Borg et al. 2014; Merten et al. 2016b). This was one of our initial motivations for the development of *IL*.

There are several possible reasons for the poor behavior of *ComL* in comparison to IL_{Com}. It is interesting that the precision of *ComL* is roughly 60% in this new study. That means, the issue IDs given by the developers are only partly correct. This observation is similar to research on developers' commit behavior and the contents of commits (Herzig and Zeller 2013; Kirinuki et al. 2014). These studies report on tangled changes, that is, a commit often comprises multiple unrelated issues. Also, we observed that developers manually excluded files in one commit, which were correct in the gold standard, and then included these files in a follow-up commit. A reason for this behavior could be a change of the requirement during the project time. Thus, the exclusion behavior was correct when the commit was performed, but was wrong for the final state of the requirement. The reasons for the lower recall of *ComL* in comparison to IL_{Com} could be select interactions. Select interactions are not detected by commits. These missed files also affect the application of source code structure-based recall improvement.

The improvement techniques developed in our last studies also proved to be reasonable in this new study. Moreover, the improvement techniques also performed well for links created with *IR* and *ComL*. By applying our wrong link detection techniques, the precision is improved, independent of how the links were created. As wrong link detection techniques

impair recall, we apply source code structure-based recall improvement. The improvement of recall by using the source code structure worked reasonable for *IL* in the last two studies and is outperformed in this new study. The application of recall improvement in this new study resulted in the best overall recall for all studies.

Altogether, we showed that the creation of links with interaction and commit data by IL_{Com_i} achieves very good precision and recall. This confirms our assumption that the additional effort of manually selecting the requirement to work on caused the low precision of *IL* in our previous study (Hübner and Paech 2018). We think that precision and recall can be even better, if developers directly use the links created during the projects, as in the Mylyn project. If the developers use the links they created , the quality of these links is more important for them and thus, we expect them to be more careful in the assignment of issue IDs to commits.

6.3 Threats to Validity

As described in our previous study (Hübner and Paech 2018), the internal validity is threatened when manual validation of trace links in the gold standard is performed by the students working as developers in a project relevant for their grades. However, this ensured that the experts created the gold standard. Also, the evaluation of the links was performed after the project had 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* techniques is a crucial factor. Regarding preprocessing, we performed all common steps including identifier splitting, which is specific to our data set used. However, the low threshold values impair the results for the precision of *IR*. Therefore, further comparison of IL_{Com} 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 the usage of the tooling by developers. The generalizability of a study based on one student project only is clearly limited. Although explicitly requested, not all commits contained a Jira issue ID in the commit messages. This affects the resulting association of recorded interaction logs to requirement issues and thus, the created trace links. However, the percentage of commits with issue IDs is similar to those reported for open source projects in Rath et al. (2017). Companies' guidelines for using issue IDs in commit messages are even more stringent than in open source projects, e.g. due to safety and regulatory concerns. This indicates that the results of our evaluation might also apply for industry projects.

7 Conclusion and Future Work

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

Our new approach and tool builds on the common practice to specify issue IDs in commit messages. It uses these issue IDs from commit messages to assign interaction log recordings to requirements. IL_{Com} has a precision of 90.0% and a recall of 79.0% which outperforms the results of our previous study (Hübner and Paech 2018) (precision of 68.2% and recall of

46.5%). Precision is not yet perfect, but we think that this is a very good basis for continuous link creation and usage. Furthermore, the new approach is applicable also in situations where developers are not particularly interested in interaction recording. We showed that our new approach outperforms *IR* and purely commit-based linking and is superior to current machine learning-based approaches (Rath et al. 2018). Clearly, it is interesting to confirm these findings in further studies and to study whether this also holds for more structured requirements where *IR* is typically used.

In our trace link maintenance SLR, we developed a generic TM process and characterized the 16 identified distinct TM approaches with this process. Overall, most of the TM approaches are either only automated in the impact detection part and then generate notifications to support the follow-up manual TM, or they use predefined rules to perform TM at least for predefined cases automatically. Only few approaches are completely automated. Further, only once a real world evaluation was performed. We used the generic TM process to discuss the integration of maintenance in ILC_{Com} using maintenance capabilities from two approaches of our SLR. Altogether, our ILC_{Com} approach has very good precision and recall and allows the integration of fully automated TM capabilities.

Our future work will be to implement the discussed TM extension of ILC_{Com} and perform an evaluation of the effects on precision and recall of continuously maintained links. Also, a study in which ILC_{Com}-created and -maintained links are continuously provided and used by developers during a project is part of our research agenda.

Acknowledgements We thank the students of the evaluation projects for their effort. We also would like to thank the reviewers for their constructive and thoughtful remarks that helped us to improve the structure and presentation of this paper. Furthermore, we thank Doris Keidel-Müller for proofreading the paper's manuscript.

Funding Information Open Access funding provided by Projekt DEAL.

References

Armbrust O, Ocampo A, Munch J, Katahira M, Koishi Y, Miyamoto Y (2009) Establishing and maintaining traceability between large aerospace process standards. In: ICSE Workshop traceability in emerging forms of software engineering, pp 36–40

Bird C, Rigby PC, Barr ET, Hamilton DJ, Germàn DM, Devanbu PT (2009) The promises and perils of mining git. In: Mining software repositories (MSR), pp 1–10

Blouin D, Barkowski M, Schneider M, Giese H, Dyck J, Borde E, Tamzalit D, Noppen J (2017) A Semi-Automated approach for the Co-Refinement of requirements and architecture models. In: 25th IEEE international requirements engineering conference workshops (REW). IEEE, pp 36–45

Borg M, Runeson P, Ardö A (2014) Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. Empir Softw Eng 19(6):1565–1616

Briand L, Falessi D, Nejati S, Sabetzadeh M, Yue T (2014) Traceability and sysML design slices to support safety inspections: A controlled experiment. ACM Trans Softw Eng Methodol (TOSEM) 23(1):1–43

Cleland-Huang J, Chang CK, Christensen M (2003) Event-based traceability for managing evolutionary change. IEEE Trans Softw Eng 29(9):796–810
Cleland-Huang J, Mäder P, Mirakhorli M, Amornborvornwong S (2012) Breaking the big-bang practice of traceability Pushing timely trace recommendations to project stakeholders. In: 20th IEEE international requirements engineering conference (RE). IEEE, pp 231–240
Cleland-Huang J, Gotel OCZ, Huffman Hayes J, Mäder P, Zisman A (2014) Software traceability: trends and future directions. In: Future of software engineering (FOSE). ACM, pp 55–69
Drivalos-Matragkas N, Kolovos DS, Paige RF, Fernandes KJ (2010) A State-based Approach to Traceability Maintenance. In: 6th ECMFA traceability workshop (ECMFA-TW). ACM, pp 23–30
Ebner G, Kaindl H (2002) Tracing All Around in Reengineering. IEEE Software 19(3)
Fockel M, Holtmann J, Meyer J (2012) Semi-automatic establishment and maintenance of valid traceability in automotive development processes. In: 2nd international workshop software engineering for embedded systems (SEES), pp 37–43
Gervasi V, Zowghi D (2014) Supporting traceability through affinity mining. In: 22nd IEEE international requirements engineering conference (RE). IEEE, pp 143–152
Ghabi A, Egyed A (2012) Code patterns for automatically validating requirements-to-code traces. In: 27th IEEE/ACM international conference on automated software engineering, pp 200–209
Herzig K, Zeller A (2013) The impact of tangled code changes. In: Mining software repositories (MSR). IEEE, pp 121–130
Hübner P., Paech B (2017) Using interaction data for continuous creation of trace links between source code and requirements in issue tracking systems. In: Requirements engineering: Foundation for software quality (REFSQ). Springer, pp 291–307
Hübner P, Paech B (2018) Evaluation of techniques to detect wrong interaction based trace links. In: Requirements engineering: Foundation for software quality (REFSQ). Springer, pp 75–91
Hübner P, Paech B (2019) Increasing precision of automatically generated trace links. In: Requirements engineering: Foundation for software quality (REFSQ). Springer, pp 73–89
Huffman Hayes J, Dekhtyar A, Sundaram SK (2006) Advancing candidate link generation for requirements tracing The study of methods. IEEE Trans Softw Eng 32(1):4–19
Kersten M, Murphy GC (2006) Using task context to improve programmer productivity. In: 14th ACM SIGSOFT international symposium on foundations of software engineering (FSE), vol 14. ACM, pp 1–11
Kleffmann M, Book M, Gruhn V (2013) Towards recovering and maintaining trace links for model sketches across interactive displays. In: 7th international workshop traceability in emerging forms of software engineering (TEFSE), pp 23–29
Kirinuki H, Higo Y, Hotta K, Kusumoto S (2014) Hey! Are You Committing Tangled Changes? In: 22nd International Conference on Program Comprehension (ICPC). ACM, pp 262–265
Kitchenham BA, Charters S (2007) Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report Version 2.3, EBSE-2007-01, School of Computer Science and Mathematics, Keele University, and Department of Computer Science University of Durham
Konôpka M, Bieliková M (2015) Software Developer Activity as a Source for Identifying Hidden Source Code Dependencies. In: Theory & practice of computer science (SOFSEM). Springer, pp 449–462
Mäder P, Egyed A (2015) Do developers benefit from requirements traceability when evolving and maintaining a software system? Empir Softw Eng 20(2):413–441
Mäder P, Gotel O (2012) Towards automated traceability maintenance. J Syst Softw 85(10):2205–2227
Maalej W, Kurtanovic Z, Felfernig A (2014) What stakeholders need to know about requirements. In: 4th international workshop on empirical requirements engineering (empiRE). IEEE, pp 64–71
Maro S, Steghöfer J (2016) Capra: A configurable and extendable traceability management tool. In: 24th IEEE international requirements engineering conference (RE). IEEE, pp 407–408
Maro S, Anjorin A, Wohlrab R, Steghöfer J (2016) Traceability maintenance: Factors and guidelines. In: 31th IEEE/ACM international conference on automated software engineering (ASE), vol 31. ACM, pp 414–425
Merten T, Falisy M, Hübner P, Quirchmayr T, Bürsner S, Paech B (2016a) Software Feature Request Detection in Issue Tracking Systems. In: 24th IEEE International Requirements Engineering Conference (RE). IEEE
Merten T, Krämer D, Mager B, Schell P, Bürsner S, Paech B (2016b) Do Information Retrieval Algorithms for Automated Traceability Perform Effectively on Issue Tracking System Data? In: Requirements Engineering: Foundation for Software Quality (REFSQ). Springer, pp 45–62
Omoronyia I, Sindre G, Roper M, Ferguson J, Wood M (2009) Use case to source code Traceability: The developer navigation view point. In: 17th IEEE international requirements engineering conference (RE), pp 237–242. IEEE

Omoronyia I, Sindre G, Stålhane T (2011) Exploring a Bayesian and linear approach to requirements traceability. Inform Softw Technol (IST) 53(8):851–871

Paige RF, Drivalos N, Kolovos DS, Fernandes KJ, Power C, Olsen GK, Zschaler S (2011) Rigorous identification and encoding of trace-links in model-driven engineering. Softw Syste Model 10(4):469

Rahimi M, Goss W, Cleland-Huang J (2016) Evolving Requirements-to-Code Trace Links across Versions of a Software System. In: IEEE International conference on software maintenance and evolution (ICSME), pp 99–109

Rath M, Rempel P, Mäder P (2017) The IlmSeven Dataset. In: 25th IEEE international requirements engineering conference (RE). IEEE, pp 516–519

Rath M, Rendall K, Guo JLC, Cleland-Huang J, Mäder P (2018) Traceability in the Wild Automatically Augmenting Incomplete Trace Links. In: 40th international conference on software engineering (ICSE). ACM, pp 834–845

Seibel A, Hebig R, Giese H (2012) Traceability in Model-Driven engineering efficient and scalable traceability maintenance. In: Software & systems traceability. Springer, pp 215–240

Schwarz H, Ebert J, Winter A (2009) Graph-based traceability: a comprehensive approach. Softw Syst Model 9(4):473–492

Soh Z, Khomh F, Guéhéneuc Y, Antoniol G (2017) Noise in Mylyn interaction traces and its impact on developers and recommendation systems. Empirical Software Engineering

Wohlrab R, Steghöfer J, Knauss E, Maro S, Anjorin A (2016) Collaborative traceability Management: Challenges and opportunities. In: 24th IEEE international requirements engineering conference (RE).. IEEE, pp 216–225

Ying P, Yong T, Xiaoping Y (2009) Software artifacts management based on dataspace. In: WASE International conference on information engineering (ICIE), vol 2. IEEE, pp 214–217