Using Interaction Logs for Creation and Maintenance of Trace Links

Paul Hübner and Barbara Paech

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

1 Problem and Motivation

We present an innovative, data-driven solution to an old requirements engineering problem [1]. Trace links between requirements and source code are beneficial for many software engineering tasks during development, e.g. maintenance, program comprehension and reengineering [2]. If trace links are created and used continuously during a project, they need to have high precision (correctness) and recall (completeness) to be useful. In addition, it is necessary to keep trace links up-to-date for the continuous usage of the links during a project. Thus, maintenance of links along with changes in linked artefact is essential [3].

To ease burden of linking for the developers, automatic linking approaches have been proposed. Most of them use the textual contents of artefacts together with information retrieval (IR) and do not consider the maintenance of links [4]. Such approaches are often intended for a once in a time retrospective application for specific purposes, such as verification that all requirements are implemented, or to justify the safe operation of a system [5]. These approaches focus on high recall and accept the burden of manual processing the automatically generated link candidates [6]. For link maintenance such approaches favour the complete regeneration of all links by simply reapplying the link creation.

In summary, existing automatic trace link creation approach are not capable to continuously provide and maintain links with high precision for direct usage by developers during a project. In practice, developers trace requirements to code often by manually linking requirements from an issue tracker system (ITS) to code in a version control system (VCS) by providing issue IDs in commit messages. However, this has typically quite low recall [7].

We therefore extended the commit linking by data captured from developers work. In our completely automatic approach IL_{Com}, we use interaction logs recorded in an IDE while a developer is working on code files to implement a requirement. To assign the recorded interaction logs automatically to the requirement worked on, IL_{Com} uses the issue IDs of requirements provided by developers in commit messages. This enables continuous link creation after each commit without manual effort from the developers.

We evaluated our approach in several studies and showed that IL_{Com} can create trace links with almost perfect precision and recall as good as IR [8]. In our presentation we introduce IL_{Com} and the results of the evaluation studies. Further we sketch how maintenance of existing links along with changes in linked artefacts can be integrated in IL_{Com}.

2 $\mathbf{IL_{Com}}$ Approach

Figure 1 shows the three steps of the IL_{Com} approach. In the first step interactions of developers with source code files are recorded in an IDE. When a developer performs a commit to a VCS like Git, and provides an issue ID in the commit message, the recorded interactions since the last commit, are assigned to the issue with this ID.

In the second step the assigned interaction logs are used to create trace links. First, interaction events from the logs are restricted to events with code files, directly triggered by the developer, and in certain IDE parts (Editor, Navigator). The files touched in these events are linked to the issue. In addition to the event type and the IDE part, each link is attributed with the frequency and duration for the link. The frequency is the number of interaction events, and the duration is the sum of all durations of interaction events used for the link.

In the third step the attributes of a link and the source code structure (SCS), i.e. references between source code files, are used to remove potential wrong links and add further potential correct links. For the removal threshold values for the attributes are used, e.g. only links with a certain duration, frequency, IDE part or interaction type. Furthermore, only code files which are also connected by SCS with each other are linked to an issue. Then SCS is used to add further links. Code files referenced by a file which is already linked to an issue are linked to that issue as well.

Table 1: Evaluation Results for IL_{Com}, *ComL* and *IR*

Approach	Precision	Recall	#Links				
			CE	TP	FP	GS	FN
IL_{Com}	0.900	0.790	271	244	27	309	65
ComL	0.675	0.443	203	137	66	309	172
IR	0.369	0.557	466	172	294	309	137

* For *IR*, the vector space model (VSM) technique with a similarity threshold of 0.2 was used

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

We compared IL_{Com} in different studies with commit based linking (*ComL*), i.e. link creation based on issue IDs from commit messages and changed files of commits, and IR. Table 1 shows the overview of the results of the last evaluation study [8]. In the study

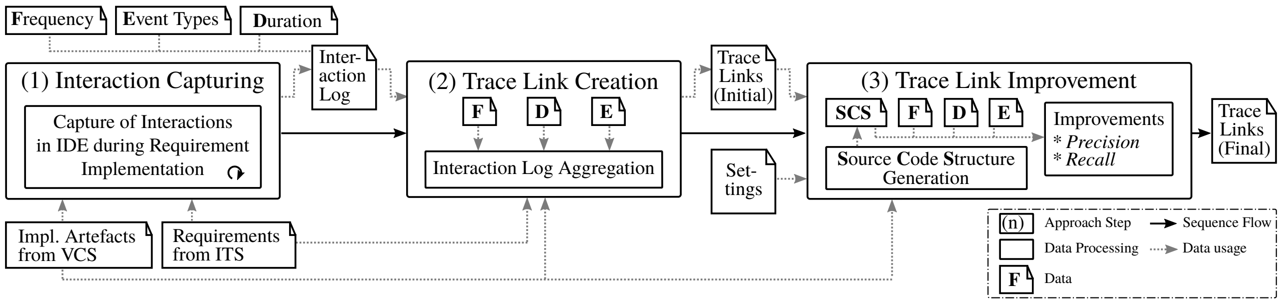

Figure 1: IL_{Com} Approach: (1) Interaction Capturing, (2) Trace Link Creation and (3) Improvement

IL_{Com} has created links with almost perfect precision of 90% and good recall of 79%. It outperformed the other link creation techniques.

3 Trace Link Maintenance

During a literature review on trace link maintenance (TM) we identified a general TM process describe in the following paragraph. The process consists of 4 steps separated in an *impact detection* and *execution part*. The *impact detection* part 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 for the next steps. The *change execution* part consists of the *determination of necessary link changes* based on the previously generated output and the *execution of those changes*.

We selected TM capabilities from two approaches identified in our literature review and used them to extend IL_{Com}. These approaches use the same artefacts as IL_{Com} and are fully automated.

The *detection of change* in IL_{Com} is performed automatically after each commit by evaluating the interaction types. A change is indicated by an edit interaction.

The *detection of impacted* links is performed by the adapted impact rules from the two TM approaches and IL_{Com}-specific rules. The rules from the approach of Ghabi and Egyed [9], use the SCS references between methods to calculate the proximity of methods to already linked requirements. For the impacted link detection an adjustable threshold for the proximity is used. The rules from the approach of Rahimi and Cleland-Huang [3], use textual differences between two artefact versions to detect certain refactorings. An example for a refactoring is *extract method to class*. In this refactoring all links referring to the class are considered as impacted. In addition in IL_{Com} the recorded interactions are used to detect refactorings. IDEs provide capabilities to perform certain refactorings, like *extract method to class*. Such refactorings are directly detected in the interactions. Furthermore, patterns of low level interactions which comprise the interaction sequence of a certain refactoring are defined and automatically detected in the interactions.

All three kind of impact detection rules output a change type. Based on that, the *determination of necessary link change* and *execution of change* is performed fully automated in IL_{Com}, i.e. change type specific rules perform the removal of existing links and adding of missing links. Links are removed/added if the proximity score of a method drops/rises below/above the specified threshold. For detected refactorings either by interactions or textual differences, refactoring specific link change rules are defined, e.g. when detecting the *extract method to class* refactoring, links are moved to the newly created class.

4 Conclusion

Our IL_{Com} approach is the first trace link creation and maintenance approach with very good precision and recall which does not require any manual work from the developers besides the common commit-based linking.

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

References

[1] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe, "Toward Data-Driven Requirements Engineering," *IEEE Software*, vol. 33, no. 1, pp. 48–54, 2016.

[2] P. Mäder and A. Egyed, "Do developers benefit from requirements traceability when evolving and maintaining a software system?" *Empir. Software Eng.*, vol. 20, no. 2, pp. 413–441, 2015.

[3] M. Rahimi and J. Cleland-Huang, "Evolving software trace links between requirements and source code," *Empir. Software Eng.*, vol. 23, no. 4, pp. 2198–2231, 2018.

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

[5] L. Briand, D. Falessi, S. Nejati, M. Sabetzadeh, and T. Yue, "Traceability and SysML design slices to support safety inspections," *ToSEM*, vol. 23, no. 1, pp. 1–43, 2014.

[6] J. Cleland-Huang, O. C. Z. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman, "Software traceability: trends and future directions," in *FOSE*, ACM, 2014, pp. 55–69.

[7] M. Rath, J. Rendall, J. L. C. Guo, J. Cleland-Huang, and P. Mäder, "Traceability in the Wild: Automatically Augmenting Incomplete Trace Links," in *ICSE*, IEEE, 2018, pp. 834–845.

[8] P. Hübner and B. Paech, "Increasing Precision of Automatically Generated Trace Links," in *REFSQ*, Springer, 2019, pp. 73–89.

[9] A. Ghabi and A. Egyed, "Code patterns for automatically validating requirements-to-code traces," in *ASE*, IEEE/ACM, 2012, pp. 200–209.