# Challenges in Using Interaction Data for Trace Link Creation

Paul Hübner and Barbara Paech

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

**Abstract.** For direct usage of trace links during a development project the precision of information retrieval (IR) based trace link creation is not sufficient [1]. Too many false links generated by IR hinder developers to directly use the resulting links and require further assessment by an expert. Thus, we develop an interaction based trace link creation approach (IL) aiming to create trace links with perfect precision. In IL we utilize the interactions of developers recorded in the IDE while working on a requirement. We already performed an evaluation of our IL approach with open source data [2]. For that we could show that IL creates trace links with perfect precision and has a better recall than IR. However, in a follow up evaluation in which we applied our approach in a student project it turned out that the precision of created trace links was not perfect. Now we are investigating the reasons for this not perfect precision, in particular we try to find techniques to detect wrong links. In this short paper we motivate and introduce our IL approach, compare the two different projects in which we applied IL, discuss reasons why links created by IL can be wrong and outline the challenges how to countervail the creation of wrong links.

## 1 Introduction

Trace link creation approaches are typically based on IR and use structured requirements and source code [1]. Due to the usage purposes of such approaches they focus on recall optimization, e.g. verification tasks like checking that all requirements from a specification are implemented after a project is finished. For such tasks it is more important to find all links than all found links being correct. If trace links are only created once, the effort to remove wrong links is acceptable. The usage purpose of interaction based trace link creation approach (IL) is different. First we apply IL in development projects with unstructured requirement data often used in issue tracking systems. Second for many development activities it is helpful to consider links between requirements and source code during development, e.g. in maintenance tasks and for program comparison [3]. Thus the main research goal for our IL approach is to optimize precision and keep recall as good as possible, so that trace links created by our approach can directly be used during a project [4].

## 2 IL Approach

Figure 1 shows the overview of the three steps IL consists of. In the first step interactions are recorded in

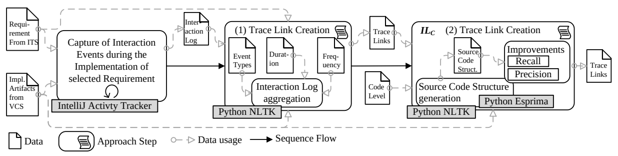

Figure 1: *IL* Approach: Interaction Capturing, Trace Link Creation and Source Code Structure Utilization

the IDE of a developer. We developed different tools for the capturing of interactions supporting Eclipse[1] and IntelliJ[2] IDE. In the second step trace links are generated based on the captured interaction logs by means of a Python NLTK[3] based tool. In the third step further trace links are added by utilizing the source code structure to improve recall. By source code structure we denote the call and data dependencies between code files and classes. Additional links are created using the source code files of trace links created in the previous step as input and then follow the source code structure till a certain level.

## 3 IL and IR Evaluation Results

Table 1: Data Sets used to evaluate IL

| Data Set | #Req. | #Interactions | #Impl. Artifcats | | #Developers |
|---|---|---|---|---|---|
| | | | Git | Touched | |
| *Mylyn 2007* ($M_{2007}$) | 50 | 111617 | 756 | 585 | 19 (3) |
| *Mylyn 2012* ($M_{2012}$) | 50 | 111318 | 2119 | 172 | 20 (3) |
| *Student 2017* ($S_{2017}$) | 19 | 1171290 | 151 | 312 | 3 |

In the first evaluation study of IL we applied IL to interaction logs recorded in the Mylyn[4] open source project. As shown in Table 1 we extracted two data sets comprising 50 requirements from the Mylyn project. Due to the large size of the project we could not create a gold standard for the trace links and thus only calculated relative recall values [5]. This was the major motivation to also evaluate IL in a student project context, in which we had the possibility to create a gold standard of trace links with the help of the project's experts. As shown in the last row of Table 1 the student project comprises 19 requirements. The large difference in the number of recorded interaction is the result of different tools used. Mylyn used Eclipse and the Mylyn plug-in itself to record interactions whereas in the student project IntelliJ and a

[1]`https://www.eclipse.org/`
[2]`https://www.jetbrains.com/idea/`
[3]`http://www.nltk.org/`
[4]`http://www.eclipse.org/mylyn/`

version of the Activity Tracker[5] IntellJ plug-in, which we extended to also record interactions with requirements, have been used. The interactions recorded within Eclipse and Mylyn are focused on direct source code and source code related interactions. In IntellJ almost all interactions are recorded, e.g. the usage of context menus and advanced actions, like committing to Git, which are often performed by special dialogs. Table 2 shows the results of applying IL and IR with

Table 2: IL and IR Results Overview

| | *Mylyn*$_{2007}$ | | | *Mylyn*$_{2012}$ | | | *Student*$_{2017}$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | $IL$ | $IR_{VSM(0.7)}$ | $IR_{LSI(0.3)}$ | $IL$ | $IR_{VSM(0.5)}$ | $IR_{LSI(0.3)}$ | $IL$ | $IR_{VSM(0.2)}$ | $IR_{LSI(0.05)}$ |
| GS Links | | 1324 | | | 491 | | | 217 | |
| Link Cand. | 1148 | 596 | 1058 | 240 | 185 | 920 | 372 | 642 | 363 |
| Correct Links | 1148 | 204 | 328 | 240 | 25 | 274 | 160 | 104 | 77 |
| Wrong Links | 0 | 392 | 730 | 0 | 160 | 646 | 212 | 538 | 286 |
| Not found Links | 176 | 1120 | 996 | 251 | 466 | 217 | 57 | 182 | 140 |
| Precision | 1.000 | 0.342 | 0.310 | 1.000 | 0.135 | 0.298 | 0.430 | 0.162 | 0.212 |
| Recall | 0.867 | 0.154 | 0.248 | 0.489 | 0.051 | 0.558 | 0.737 | 0.479 | 0.355 |
| $F_{0.5}$ | 0.970 | 0.275 | 0.295 | 0.827 | 0.102 | 0.328 | 0.469 | 0.187 | 0.231 |

two common IR algorithms vector space model (VSM) and latent semantic indexing (LSI) [1] to the three data sets. Due to the different characteristics of the requirement data in the different data sets (i.e. the text of the requirements in the student project where much shorter as in Mylyn) the IR methods required different thresholds as shown in the table.

When looking at the results of IL it can be seen that its precision is perfect for the two Mylyn data sets but only at 43.0% for the student project. Recall of IL is acceptable for our concern and could be improved in all cases when using source code structure to find more links. For the overall rating of the achieved results we choose $F_{0.5}$ measure to weight precision twice as much as recall. So that $F_{0.5}$ mainly correspond to the precision results for all data sets. When looking at the $F_{0.5}$ measures it can be seen that IL outperforms IR in all three data sets.

However, the precision of IL in the student data set is not suitable for direct usage of the resulting trace links, since at least every second resulting trace link would be wrong. Thus we started to investigate in the reasons for that bad precision and how to countervail it. The main reason are wrong interactions caused by developers interacting with implementation artifacts not contributing to the actual requirement. For the interaction recording the developers had to manually activate the requirement they are working on. In contrast to the student developers the Mylyn developers did this in very strict and disciplined manner so that no wrong interactions were recorded.

## 4 Interaction Data Challenges

We plan to detect interactions resulting in potential wrong links. For this we plan to utilize the data recorded in the interaction logs, that is the duration, the frequency for implementation artifacts, the part of the IDE in which the interaction occurred and the type of interaction.

[5]`https://plugins.jetbrains.com/plugin/8126-activity-tracker`

Our hypothesis is that interactions with longer duration or higher interaction frequencies are more likely to result in correct trace links. For the part of the IDE, the editor and navigator seem the most valuable IDE parts. For interaction types edit seem to be more important than selects.

Based on a preliminary analysis of the interaction logs we also want to investigate source code related wrong link detection techniques. In our already performed analyses we found that there are implementation artifacts all developers interacted with during the implementation of all requirements. Moreover there are also artifacts the developers did not interact with but they are contained in the gold standard of trace links. It also seems to be promising to use the code structure for precision based improvements as source code artifacts used within the implementation of one requirement should be linked by the code structure with each other. Thus, not connected implementation artifacts could be removed.

The basic idea of analyzing the code structure of files involved in the implementation of a requirement is also part of source code structure analysis research [6]. We will also investigate in this research direction to find more source code related metrics supporting the detection of code not relevant for trace link creation. In addition to just applying a single wrong link detection technique we also plan to investigate in the combinations of these techniques.

These techniques could be used during the creation of trace links so that on each commit to the version control system the developer gets the list of link candidates produced by IL since the last commit and can rate the links supported by our wrong link detection techniques introduced in this paper.

## References

[1] 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. 1–52, may 2013.

[2] P. Hübner and B. Paech, “Using Interaction Data for Continuous Creation of Trace Links Between Source Code and Requirements in Issue Tracking Systems,” in *REFSQ*, vol. LNCS 10153. Springer, 2017, pp. 291–307.

[3] 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, apr 2015.

[4] P. Hübner, “Quality Improvements for Trace Links between Source Code and Requirements,” in *REFSQ Doctoral Symp.*, vol. 1564. CEUR-WS, 2016.

[5] M. Fricke, “Measuring recall,” *Journal of Information Science*, vol. 24, no. 6, pp. 409–417, 1998.

[6] K. Herzig, S. Just, and A. Zeller, “The impact of tangled code changes on defect prediction models,” *Empir. Software Eng.*, vol. 21, no. 2, pp. 303–336, Apr 2016.