State of the Art of Traceability in Open-Source Projects

Michael Rath, Maksim Goman, Patrick Mäder
Software Engineering for Safety-Critical Systems
Technische Universität Ilmenau
{michael.rath, maksim.goman, patrick.maeder}@tu-ilmenau.de

Abstract

Software and systems traceability is widely accepted as an essential element for supporting many software development tasks. Todays version control systems provide inbuilt features that allow developers to tag each commit with one or more issue IDs, thereby providing the building blocks from which project-wide traceability can be established between feature requests, bug fixes, commits, source code, and tests. We analyzed seven large open source projects to investigate to what extent developers explicitly established traces between issues and commits. Therefore, we categorized resolved issues and commits and studied the traces between the resulting artifact clusters. Among other metrics, our research shows, that 70% of all resolved issues are linked to commits. However, in the opposite direction, only 48% of the commits are linked to issues. Thus, open-source developers actively establish traceability. Nevertheless, automated traceability techniques might increase the amount of interlinking.

1 Introduction

Traceability provides support for many different software engineering activities including change impact analysis, test regression selection, safety analysis, and coverage analysis [1]. The interlinking of development artifacts from issue trackers like JIRA and version control systems for storing the source code like Git play an important role in software development, during the release planning or the bug triaging. In open-source systems, it has become common practice for developers to tag commits with issue IDs. In large projects, such as the ones from the Apache Foundation, this procedure is even reflected in the guidelines, which state that *"You need to make sure that the commit message contains at least [...] a reference to the Bugzilla or JIRA issue [...]"*[1]. Creating such tags establishes explicit links between commits and issues, such as feature requests, improvements and bug reports. Such a link allows to trace from an issue to the files modified in order to resolve the issue. Furthermore, leveraging this information enables to build rich trace graphs [2, 4] supporting the for mentioned software engineering activities. The explicit links form issue tracker artifacts to source code files are also the foundation for more advanced trace techniques towards classes or even methods in the source code.

In this paper, we briefly investigate the existing interlinking from issue tracker systems to source code in order to draw conclusions about traceability completeness. Two scenarios are considered. The first deals with the issue to commit perspective and shows the number of issues linked to commits. The second is the inverse direction from commits to issue uncovering non-linked commits. For our research, we make use of an existing dataset containing the necessary development artifacts and traces between issues and commits collected from seven large open-source systems.

2 Data Model

We used a recently published dataset [5] to study the existing traces from issue tracker systems to source code. It contains issues, commits and trace information extracted from seven large open-source systems from different domains like programming languages, databases, and web servers and thus covers a wide range of applications. For a structured analysis, we first categorized the issues and commits as follows.

In JIRA, the issues are typed to allow adaptation to different phases in software development. We propose five issue categories based on this issue type information: *Feature* (i. e. a new feature of the product), *Improvement* (i. e. an enhancement to an existing feature), *Task* (i. e. a task that needs to be done), *Bug* (i. e. a problem which impairs or prevents the functions of the product). The fifth category, *Other*, collects all issue types not matching any of the four specific categories and includes issue types such as *Patch*, *Release*, or *Question*. Additionally, we only considered finished work, i. e. the *status* of an issue needs to be "Resolved" or "Closed" and its *resolution* needs to be "Fixed" or "Done".

In the Git version control system, changes are organized as atomic commits. A commit represents a set of modified files. We categorized commits based on these contained files. The file extensions and file paths in the project directory layout[2] were used to

[1] http://www.apache.org/dev/committers.html#applying-patches

[2] http://maven.apache.org/guides/introduction/

distinguish three different file types: source code files (i. e. the code of the product written in a programming language), test files (i. e. tests written in a programming language), and admin files (neither source code nor test files such as files required to build the software). Based on the file types, we categorized commits as *Source*, *Test*, and *Admin* if they only contain files of the respective file type. Additionally, we added the category *Source & Test* for commits consisting of source code and test files. The rational for this category is that developers might bundle changes to source code files and the corresponding tests in the same commit. The last category, *Mixed*, is used for commits containing all three file types.

3 Analysing Existing Trace Links

We analyzed the number of existing traces between issues and commits for each project. The analysis was done by running multiple SQL queries utilizing the database interface of the dataset.

At first, we focused on traces from issues to commits. For each issue, the number of traces to commits were quantified as *Linked 1:1*. i. e. the issue is traced exactly to one commit, *Linked 1:n*, i. e. the issue is traced to more than one commit, and *Non-linked*, i. e. there is no explicit trace from the issue to any commit. The aggregated results for the issue categories are reported in Table 1. For example, of 1354 issues categorized as improvement in project DERBY, 562 issues are linked exactly to one commit, 451 improvements are linked to more than one commit and 341 improvements are not linked to any commit.

Table 2 depicts the number of traces from the perspective of commits. The results are aggregated by link type and the introduced commit type categories. Again, we analyzed the distribution of 1:1 links, 1:n links and non-linked commits. For example, in project DERBY, 1885 commits fall into category *Test*, out of which 1497 commits are linked to exactly one issue, 105 commits are linked to more than one issue, and 283 commits are not linked to any issue.

4 Discussion

Issue to commit traces. Table 1 reveals, the percentage of linked issue varies across the different projects, reaching a maximum of 82% for project INFINISPAN and a minimum of 60% in project GROOVY. Issues belonging to category *Task* are consistently the least linked issues (neglecting the fallback type *Other*), while *Improvements* are the most linked, followed by *Bugs*.

Another insight of Table 1 is that linking an issue to multiple commits is a common practice across the studied projects independent of the issue category. Thus, numerous file modifications are made until an issue is eventually resolved. E. g. in project DROOLS, roughly half of the features linked to commits are implemented in this manner. Across all project, the percentage of link type *Linked 1:n* varies from 10% (PIG) to 26% (DERBY) of all resolved issues.

The analysis shows, that the majority of issues are linked to commits. However, still there are non-linked issues. This is especially interesting, since this study only considered issues, that represent finished work according to the development process. Hence, the resolution of the non-linked issues did not require modifications of any file in the version control system. This may be true for a small amount of issues. However, the number of non-linked issues indicates, that the developers violated the project guidelines and omitted trace links. Thus, (semi-)automated trace recovery techniques, e. g. [3, 7], are required to create links for all non-linked issues. Furthermore, since an issue might be linked to multiple commits, these techniques should be also applied to already linked issues.

Commit to issue traces. Again, the first observation depicted in Table 2 is that not all commits are linked to issues. In project MAVEN, only 24% of the commits are explicitly linked to issues. The highest linkage from commits to issues is achieved in project PIG with 93%. Across all projects, the category of *Admin* commits is the one with the lowest linkage ranging from 11% to 82%. However, in the context of traceability analysis from issues to code, this category is of minor importance. Most commits belong to categories *Source*, *Test*, and their combination *Source & Test*. Their link percentage greatly varies among the studied projects. In projects MAVEN and SEAM2, roughly only a third of the commits belonging to these categories are linked to issues. The highest percentage in this respect is achieved in project DERBY, where more than 80% of source and test commits are linked.

In contrast to the issue to commit perspective, Table 2 shows, that link type *Linked 1:n* (i. e. one commit addresses multiple distinct issues) is not that common. In project DERBY, 4.8% of all commits are linked to multiple issues. For the other projects, this value is below 2% of all commits of the project.

Despite projects PIG and SEAM2, it is also common behavior to bundle changes of source code files and test files in the same commit. This may indicate, that the developers indeed directly altered or adapted the test cases for the modified source code files.

This analysis shows, that in some projects a large number of commits are not explicitly linked to issues. On reason may be, that the developers only tag one commit out of a series of commits when resolving an issue, as described in [6]. Trace recovery techniques should focus on non-linked commits to increase trace completeness, since the *Linked 1:n* type is a minor phenomenon.

`introduction-to-the-standard-directory-layout.html`

Table 1: Number of existing traces from resolved issues to commits categorized by issue types and link types for the studied projects. Bold values indicate the highest percentage of linked issues for each project.

		Project						
Link Type	Issue Category	Derby	Drools	Groovy	Infinispan	Maven	Pig	Seam2
Linked 1:1	Bug	1,470	1,032	1,668	1,932	715	1,333	993
	Feature	0	216	117	362	44	111	377
	Improvement	562	57	475	409	282	359	7
	Task	227	74	70	518	74	146	251
	Other	0	14	17	186	17	35	69
Linked 1:n	Bug	558	354	540	443	152	191	209
	Feature	0	265	60	172	26	31	119
	Improvement	451	29	212	123	76	95	1
	Task	191	60	30	146	18	40	59
	Other	0	15	7	27	2	1	18
Non-Linked	Bug	610	455	1429	506	621	545	338
	Feature	0	208	107	123	77	54	290
	Improvement	341	33	334	88	229	148	1
	Task	194	205	166	211	202	343	128
	Other	0	28	53	42	23	11	16
Linked [%]	Bug	**77**	**75**	61	82	58	74	78
	Feature	-	70	62	81	48	72	63
	Improvement	75	72	**67**	**86**	**61**	**75**	**89**
	Task	68	40	38	76	31	35	71
	Other	-	51	31	84	45	77	84
	Overall	75	69	60	82	55	68	73

Table 2: Number of existing traces from commits to resolved issues categorized by commit types and link types for the studied projects. Bold values indicate the highest percentage of linked commits for each project.

		Project						
Link Type	Commit Category	Derby	Drools	Groovy	Infinispan	Maven	Pig	Seam2
Linked 1:1	Admin	808	389	304	789	422	572	1,218
	Source	1,617	1,332	1,345	1,813	1,050	95	1,007
	Test	1,497	507	367	850	61	82	114
	Source & Test	880	1,472	2,198	1,663	238	57	58
	Mixed	1,552	1,191	311	1,044	582	1,965	445
Linked 1:n	Admin	41	12	4	14	20	13	28
	Source	79	38	41	26	29	2	31
	Test	105	12	9	34	0	3	1
	Source & Test	54	48	94	42	4	3	3
	Mixed	116	80	14	32	39	44	21
Non-Linked	Admin	438	1,591	1,479	749	3,511	125	4,142
	Source	413	1,385	2,891	954	2,411	24	2,139
	Test	283	851	1,204	565	353	21	670
	Source & Test	21	861	1,303	276	335	7	122
	Mixed	199	1,058	785	242	1,045	40	1,264
Linked [%]	Admin	66	20	17	52	11	82	23
	Source	80	50	32	66	31	80	**33**
	Test	85	38	24	61	15	80	15
	Source & Test	**98**	**64**	**64**	**86**	**42**	90	**33**
	Mixed	89	55	29	82	37	**98**	27
	Overall	83	47	38	69	24	93	26

5 Conclusion

In this paper, we briefly investigated the issue to commit interlinking of seven large open-source systems. The results show, the developers predominantly create explicit trace links between the development artifacts and thus follow the project guidelines. However, still a significant amount of issues in the studied projects is not linked to commits and vice versa. Out of the 27,099 resolved issues, 70% are linked to commits. The other way around, only 48% of the 64,788 investigated commits are linked to issues. Furthermore, the percentage of linked artifacts greatly varies depending on the nature of the artifact, i. e. issue type and commit type. Leveraging open-source projects for developing advanced trace recovery techniques seems promising, by studying existing traces to infer the omitted ones.

Acknowledgment

We are funded by the German Ministry of Education and Research (BMBF) grants: 01IS14026A, 01IS16003B, by DFG grant: MA 5030/3-1, and by the EU EFRE/Thüringer Aufbaubank (TAB) grant: 2015FE9033.

References

[1] J. Cleland-Huang, O. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman. Software traceability: Trends and future directions. In *Proc. FOSE*. ACM Press, 2014.

[2] M. Goman, M. Rath, and P. Mäder. Lessons Learned from Analyzing Requirements Traceability using a Graph Database. *Softwaretechnik-Trends*, 37(3), 2017.

[3] C. McMillan, D. Poshyvanyk, and M. Revelle. Combining textual and structural analysis of software artifacts for traceability link recovery. In *ICSE Workshop on Traceability in Emerging Forms of Software Engineering*, 2009.

[4] M. Rath, D. Akehurst, C. Borowski, and P. Mäder. Are graph query languages applicable for requirements traceability analysis? In *22nd International Conference on Requirements Engineering: Foundation for Software Quality REFSQ)*, 2017.

[5] M. Rath, P. Rempel, and P. Mäder. The IlmSeven Dataset. In *Requirements Engineering Conference (RE), 2017 IEEE 25th International*. IEEE, 2017.

[6] G. Schermann, M. Brandtner, S. Panichella, P. Leitner, and H. C. Gall. Discovering loners and phantoms in commit and issue data. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension, ICPC*, 2015.

[7] T. M. Tomova, M. Rath, and P. Mäder. Preprocessing Texts in Issue Tracking Systems to improve IR Techniques for Trace Creation. *Softwaretechnik-Trends*, 37(3), 2017.