Request for Comments: Conversation Patterns in Issue Tracking Systems of Open-Source Projects

Michael Rath
DLR Institute of Data Science / Technical University Ilmenau
Jena / Ilmenau, Germany
michael.rath@tu-ilmenau.de

Patrick Mäder
Technical University Ilmenau
Ilmenau, Germany
patrick.maeder@tu-ilmenau.de

ABSTRACT

Issue tracking systems play an important role in developing software systems. They provide a central place to store and maintain different development artifacts. Various studies are concerned with the contained bug reports, features, the relations among them and traces to the projects code base. However, an issue tracker can also be used as a communication channel between project contributors by attaching comments to issues. Less is known on how users actually utilize this functionality. In this paper, we study more than 270,000 comments from twelve open-source projects. We analyze to what extend comments are used and then study the structure occurring in threads of comments. Based on the order of comments and participating contributors, we identified three patterns of conversation: monolog, feedback, and collaboration. Our results show that most conversations are collaborations among two or more developers discussing the issue.

KEYWORDS

Human Factors, Issue Tracking Systems, Comments, Developer Communication

ACM Reference Format:
Michael Rath and Patrick Mäder. 2020. Request for Comments: Conversation Patterns in Issue Tracking Systems of Open-Source Projects. In *The 35th ACM/SIGAPP Symposium on Applied Computing (SAC '20), March 30-April 3, 2020, Brno, Czech Republic.* ACM, New York, NY, USA, 4 pages.

1 INTRODUCTION

Issue tracking systems (ITS) are widely used in the development process of commercial and open-source systems (OSS) [19]. ITS enable the development team to record and track the status of *issues* in the project [12], with issue being one of the basic artifacts managed by the system. The trackers replace a variety of dedicated tools used for elicitation and analysis of development artifacts, and provide a shared environment where team members can ask for advice and share opinions useful for maintenance activities or design decisions [9].

Communication among developers is vital, because of the important role ITS play in the development and the large amount of data stored there. A relevant part of an issue, e. g. a feature request, is its textual description written by the creator (issue reporter). This information alone might not be sufficient to implement and resolve the particular requirement in code. Social interaction is necessary to clarify technical aspects and resolve organizational dependencies. The documentation [4] of a prominent ITS states that "comments are a useful way to record additional details about an issue and to collaborate with team members". To address these needs, ITS allow to attach comments on issues and therefore serve as a communication channel [1] in addition to other systems, such as mailing lists [17] and chat systems [18]. However, the issue tracker is more suited for this purpose and allows to provide information in the context of artifacts, i. e. the respective issue. Together, the issue and its comments (*comment thread*) provide a consolidated view of the entire issue history.

While issue artifacts are extensively studied in the field of bug localization [21, 22], feature localization [11], feature request detection [8] and requirements traceability [15, 16], less is known on how comments are used in this regard. Murgia et al. [9] identified comments as potential data source for emotion mining. In [10], the authors state that stakeholders use comments to discuss issues and to amend technical details. Ko et al. investigated the influence of comments on issue resolution [6] and making design decisions [7]. Both studies only involve a small number of bug reports. Knauss et al. [5] analyzed about 1,200 comments of one project to identify patterns in requirements discussions. They identified a set of six requirement clarification patterns. Our work is on a much larger scale and does not focus on a particular artifact type.

In this explorative study, we investigate how comments are utilized in twelve OSS. We hereby intentionally solely focus on textual data available in the respective ITS, and don't consider communication outside of them, which already were targeted in multiple studies [2, 17]. Therefore, we study the structure of contained comment threads. Specifically, we try to answer the following research questions:

RQ-1 Who participates in ITS comment threads?
RQ-2 Which conversation patterns exist in comment threads?

We found three conversation patterns, whereas collaboration among multiple users is the dominant one.

2 METHODOLOGY

After motivating our study with an example, we define concepts used throughout this paper, and introduce our dataset.

Example thread of comments. Figure 1 shows an example of an issue[1] with comments retrieved from the issue tracker of the database project Derby. The box on top represents the issue typed as improvement. The issue's main properties are a short summary and

[1] DERBY-6752: https://issues.apache.org/jira/browse/DERBY-6752

Issue

Improvement DERBY-6752
Reporter: **Alice** Assignee: **Bob**
Created: 30/Sep/14 19:40 Resolved: 31/Jul/16 19:26
Summary: AutoloadedDriver tries to load a non-existent class, AutoloadedDriver40
Description: This happens in a static initialization block at the start of AutoloadedDriver. Is this dead code?

Comments

#1 Comment by **Carol** 01/Oct/14 21:02
It might be dead code, Alice. Work for DERBY-1984 as part of revision 1494482 seems to have removed AutoloadedDriver40.

#2 Comment by **Bob** 31/Jul/16 00:08
I attached the patch for this issue. All tests are clean.

#3 Comment by **jira-bot** 31/Jul/16 19:25
Commit 1754680 from Dave in branch 'code/trunk' [https://svn.apache.org/r1754680]
DERBY-6752: Derby tries to load non-existent class AutoloadedDriver40
This patch was contributed by Bob *[…rest of commit message]*

#4 Comment by **Dave** 31/Jul/16 19:26
Looks great on my system, too. Thanks for cleaning this up! Marked fixed.

Figure 1: Example of improvement DERBY-6752 with thread of comments showing collaboration among the developers.

a longer textual description. Additionally, meta data exists about participating users[2]: Alice, the *reporter* of the issue and Bob, the *assignee*, i. e. the developer responsible for resolving the issue. Each of the four responding *comments* has three properties: the author (*commenter*), a timestamp of creation, and the comment text. The sequential list of comments ordered by date resembles a *thread*, which we also denote as *conversation*. In the example, two additional users Carol and Dave are commenting, next to the issue's assignee Bob. There is also an automatically generated comment by the non-human contributor `jira-bot`.

Studied Contributors. The issue reporters, issue assignees and all commenters are *contributors* and usually human. In this case, we denote them as *users*. However, non-human contributors – *bots* – also comment (see Section 3), but do not report issues or are assigned to handle them. A *(project) developer* is a user with project specific knowledge capable to resolve the respective issue. Therefore we assume all assignees are developers.

Studied Sets of Issues. To systematically study comments, we divide the set of all project's issues denoted by I_{all} into several subsets (see Fig. 2). The subset I_{com} contains all issues that are commented and thus provides insights if and to what extend the comments are used in the respective project. We further divide this set in three distinct subsets: issues that are only commented by one user (I_{self}, **monolog**), ones where a user provides **feedback** (I_{feed}), and those where multiple users are **collaborating** (I_{coll}). I_{coll} is further divided based on participating users: the issue reporter $I_{coll,R}$, the issue assignee $I_{coll,A}$, or also additional other users $I_{coll,O}$. At last, comments are also programmatically created

[2]We replaced names with generic ones to respect privacy.

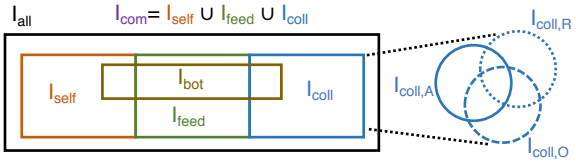

Figure 2: Venn diagram with set of all issues I_{all} of a project, as well as different studied subsets.

Table 1: Overview of the dataset

	Contributors		Issues		Com-	Per Issue†			
Project	Dev[1]	Users	$\lvert I_{all}\rvert$	$\frac{\lvert I_{com}\rvert}{\lvert I_{all}\rvert}$	ments	$\bar{C}$	max C	$\overline{Cr}$	max Cr
DERBY	125	1,009	6,969	0.95	59,829	8.9	177	2.8	23
DROOLS	69	1,402	5,103	0.67	9,763	2.8	38	1.6	13
GROOVY	65	2,677	8,137	0.84	25,345	3.7	52	2.0	26
INFINI.[2]	86	620	8,422	0.54	17,298	3.8	92	1.7	10
LOG4J2	9	1,029	2,144	0.92	13,335	6.9	125	2.5	21
MAVEN	61	2,659	5,073	0.87	17,298	3.8	307	2.3	51
PIG	337	1,038	5,234	0.90	29,896	6.3	133	2.6	24
REST.[3]	33	829	1,649	0.70	4,565	4.0	55	1.9	15
SEAM2	52	1,414	5,031	0.80	12,574	3.0	48	1.8	15
TEIID	27	310	4,899	0.96	16,808	3.5	45	1.6	8
WELD	47	527	2,518	0.65	5,341	3.2	44	1.8	10
WILDFLY	340	3,988	24,566	0.70	59,719	3.5	83	1.8	17
			79,745	0.79	271,791				

[1]Developers, [2]INFINISPAN, [3]RESTEASY, †mean and max number of comments C and distinct commenters Cr per issue

by *bots* and may occur in any conversation. Thus, I_{bot} intersects all aforementioned sets.

Study Setup. For our quantitative analysis, we obtained and preprocessed data from 12 open-source projects. We chose Atlassian Jira [3] as issue tracker, because it is the most widely used ITS tool [10]. Searching for projects, we considered two major publicly available hosting providers: the Apache Software Foundation and JBoss. We aimed for projects containing a substantial number of issues and comments. We selected seven projects previously used in studies [14] and added five comparable projects (regarding number of issues and domains). The Jira platform offers a RESTful web service, which allows to interact with the system programmatically. We implemented a collector utilizing the provided application programming interface (API). For every project we downloaded the issues and the attached comments for our analysis. Every issue in Jira has a *type* property which allows to model and distinguish different artifacts. Jira has a predefined list of types containing *bug*, *feature*, *improvement*, and *task*. The majority of issues use one of these types. However, new issue types can be configured as needed for a project. We mapped these to a fifth type denoted as *other*.

Data Demographics. Table 1 gives an overview of the dataset. It lists the number developers, as a subset of all users, the number of issues and comments, the average ($\bar{C}$) and maximum number of comments (max C) and the average ($\overline{Cr}$) and maximum number of commenters (max Cr) per issue. For example, project DERBY has 6,969 issues assigned to 125 developers, overall 59,829 comments and nearly all issues (95%) are commented. An average comment

thread contains 8.9 comments ($\bar{C}$) posed by 2.8 distinct commenters ($\overline{Cr}$). The longest thread contains 177 comments and the maximum number of users in a Derby comment thread is 23. The ratio of commented issues varies per project and is lowest for project Infinispan with about 54%. 66% of these uncommented issues in the project are resolved by the person which also reported this issue. Thus there seems no reason the comment. This pattern is also found in projects Drools, Resteasy, and Weld. However, the majority of issues (80% on average per project) is commented and each issue has on average 4 comments.

3 WHO IS COMMENTING?

In this section we take a closer look at the contributors discussing an issue and define three patterns of conversations. Due to space limitations, only a few results tables are included here. Additional results, especially the pattern distribution per issue type across the projects, are part of our published dataset [13].

3.1 Non-Human commenters (Bots) I_{bot}

In the last years, *bots* started to play an important role in many software development contexts [20]. They are used to support developers in making decisions and for coordination by aggregating information from other tools used by the project, such as the version control system, and put it in the comment thread. Further they automate repetitive tasks in software development (see third comment in Fig. 1). Our shared dataset contains further details about the bots we identified in the projects [13]. In the remaining part of the paper all comments created by bots are excluded as we focus on conversations of the developers in the projects.

3.2 Self-commented Issues (Monolog) I_{self}

By taking a closer look at our data, we found many issues involve only one person. Thus, the author of the issue (reporter), the person responsible to resolve it (assignee) and all users commenting on the issue are the same person. We denote this behavior as *self commented issue (monolog)*, visualized on the left in Figure 3. The proportion of monologs per project are shown in Table 2 column $\frac{|I_{self}|}{|I_{com}|}$. On average, one out of six (15%) of all commented issues are monologs. The practice of self commenting is present independent of the issue type. In our dataset they mostly occur in issues typed as tasks (cf. [13]). Task often have a well defined goal and thus can be handled directly by the developer without further communication.

3.3 Discussed Issues (Collaboration) I_{coll}

In this section we focus on comment threads, where users are collaborating with each other. We define *collaboration* is present in a comment thread when at least two distinct users interact in specific patterns. Considering four users U_R (issue reporter), U_A (issue assignee), and $U_{O1} \neq U_{O2}$ (two other users), we differentiate three types of collaboration:

(1) *With reporter* $I_{coll,R}$: U_R comments *after* U_{O1} commented the issue, with $U_R \neq U_{O1}$. This also covers that the other user is the assignee, i. e. $U_{O1} = U_A$.

Table 2: Proportion of commented issues I_{com} with monolog I_{self}, feedback I_{feed} and collaboration I_{coll} threads.

Project	Mono. $\frac{\lvert I_{self}\rvert}{\lvert I_{com}\rvert}$	Feed. $\frac{\lvert I_{feed}\rvert}{\lvert I_{com}\rvert}$	Collab. $\frac{\lvert I_{coll}\rvert}{\lvert I_{com}\rvert}$	Collaboration Details $\frac{\lvert I_{coll,R}\rvert}{\lvert I_{com}\rvert}$	$\frac{\lvert I_{coll,A}\rvert}{\lvert I_{com}\rvert}$	$\frac{\lvert I_{coll,O}\rvert}{\lvert I_{com}\rvert}$
DERBY	0.10	0.23	0.67	0.27	0.19	0.44
DROOLS	0.22	0.48	0.30	0.09	0.24	0.05
GROOVY	0.08	0.46	0.46	0.12	0.25	0.21
INFINISPAN	0.18	0.52	0.30	0.13	0.17	0.10
LOG4J2	0.19	0.25	0.56	0.29	0.19	0.32
MAVEN	0.11	0.45	0.44	0.12	0.16	0.27
PIG	0.03	0.31	0.66	0.34	0.14	0.35
RESTEASY	0.08	0.50	0.43	0.13	0.32	0.13
SEAM2	0.13	0.49	0.38	0.14	0.21	0.15
TEIID	0.36	0.34	0.30	0.10	0.20	0.06
WELD	0.15	0.47	0.38	0.14	0.24	0.13
WILDFLY	0.17	0.44	0.39	0.14	0.21	0.16
Average	0.15	0.41	0.44	0.17	0.21	0.20

(2) *With assignee* $I_{coll,A}$: U_A comments *after* U_{O1} commented the issue, with $U_A \neq U_{O1}$.

(3) *Among Others* $I_{coll,O}$: At least two other users besides the reporter and assignee are commenting, i. e. $\exists\, U_{O1}, U_{O2} \in \{\text{all commenters}\} \setminus \{U_A, U_R\}$.

Figure 3 illustrates typical examples for each collaboration type. The introduced collaboration types are overlapping (see Fig 2). For example there might be collaboration with the reporter as well as among other users co-occurring in the same conversation. This is the case in the motivating example shown in Fig 1, where Bob (reporter) posts a comment after Carol (other user 1) and later Dave (other user 2) also comments. On average, half (44%) of all commented issues exhibit collaboration (see Table 2) and thus this is the dominant type of conversation in the dataset. Regarding the discussed issue type (cf. [13]), collaboration is the most used form of conversation (41% on average). Improvements and (49%) and bug reports (46%) have the highest share of collaboration.

3.4 Responses on Issues (Feedback) I_{feed}

We denote conversations which are neither monologs nor collaborations as *feedback*. There, a user different from the reporter comments on the issue, which differentiates feedback from monolog. As only one user comments, there is no form of interaction in the thread (see Figure 3). For example, feature `RESTEASY-1431`[3] summarized with *"Add SNI support for netty4 integration"* gives a concrete example. There, a user different from assignee and reporter comments *"I've merged your PR, thanks."*, providing feedback for the assignees pull request (PR). Next to collaboration, feedback is on average the second most (41%) conversation type in our dataset (see Table 2). In regard to the commented issue, feedback is used in 40% of conversations independent of the issue type (cf. [13]). However, conversations about features and bug reports are more often categorized as feedback ($\geq$ 40%), than conversations about any other issue type.

[3] https://issues.jboss.org/browse/RESTEASY-1431

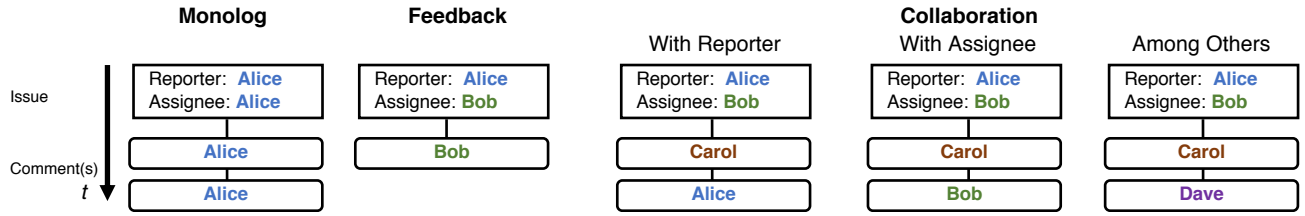

Figure 3: Types of conversation along with one example found in comment threads.

Key Takeaway (RQ1) Next to *humans*, comment threads are enriched by *bot* generated messages to bundle further information about an issue such as implementation activities or test statuses.
Key takeaway (RQ2) *Collaborations* are the most prominent form of conversation. Improvements require collaboration with the issues reporter, wheres bug reports require collaboration with the assignee. *Feedback* is the second most conversation type. It has no clear prevalence for a specific issue type, and is present in 40% of all issue conversations. Last, in *monologs* all comments are given by the reporter of the issue, which is also the assignee. This pattern is more of found in tasks rather than in bug reports.

4 CONCLUSION AND FUTURE WORK

Open source users utilize issue tracking systems (ITS) to store and maintain development artifacts such as bug reports and features. The ITS also serves as communication channel among the users, as it allows to create comments on the contained issues. Our study focuses on how users of open-source systems (OSS) use these comments for their work. We analyzed over 269,000 comments mined from 12 open-source projects. Besides humans, comments are also generated by robots aggregating additional information collected from various data sources such as the projects' version control system. By defining three conversation patterns (monologs, feedback, and collaboration), we studied the communication structure in comment threads. Our results show that collaboration, i. e. multiple users work together in order to resolve the issue, is used the most. In a future work we will focus on the content and topics discussed in the identified conversation patterns.
Our dataset along with additional material is available online [13].

Acknowledgement We are funded by the DFG grant MA 5030/3-1, the DLR grants D/943/67258261 and D/943/67262000, and BMBF grant 01IS18074E.

REFERENCES

[1] Dane Bertram, Amy Voida, Saul Greenberg, and Robert Walker. 2010. Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams. In *Proc. of the 2010 ACM conference on Computer supported cooperative work*.
[2] Swapna Gottipati, David Lo, and Jing Jiang. 2011. Finding relevant answers in software forums. In *26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*.
[3] JIRA 2018. JIRA. https://www.atlassian.com/software/jira. (2018).
[4] JIRA Documentation 2018. Commenting on an Issue. https://bit.ly/2CXZRaQ. (2018).
[5] Eric Knauss, Daniela E. Damian, Germán Poo-Caamaño, and Jane Cleland-Huang. 2012. Detecting and classifying patterns of requirements clarifications. In *2012 20th IEEE International Requirements Engineering Conference (RE)*.
[6] Andrew J. Ko and Parmit K. Chilana. 2010. How power users help and hinder open bug reporting. In *Proc. of the 28th Int. Conf. on Human Factors in Computing Systems, CHI*.
[7] Andrew J. Ko and Parmit K. Chilana. 2011. Design, discussion, and dissent in open bug reports. In *iConference 2011, Inspiration, Integrity, and Intrepidity*.
[8] Thorsten Merten, Matus Falis, Paul Hübner, Thomas Quirchmayr, Simone Bürsner, and Barbara Paech. 2016. Software Feature Request Detection in Issue Tracking Systems. In *24th IEEE Int. Requirements Engineering Conf., RE*.
[9] Alessandro Murgia, Parastou Tourani, Bram Adams, and Marco Ortu. 2014. Do developers feel emotions? an exploratory analysis of emotions in software artifacts. In *11th Working Conference on Mining Software Repositories, MSR*.
[10] Marco Ortu, Giuseppe Destefanis, Bram Adams, Alessandro Murgia, Michele Marchesi, and Roberto Tonelli. 2015. The JIRA Repository Dataset: Understanding Social Aspects of Software Development. In *Proc. of the 11th Int. Conf. on Predictive Models and Data Analytics in Software Engineering, PROMISE*.
[11] D. Poshyvanyk, Y.-G. Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich. 2007. Feature Location Using Probabilistic Ranking of Methods Based on Execution Scenarios and Information Retrieval. *IEEE Trans. Softw. Eng.* (2007).
[12] Roger S Pressman. 2005. *Software engineering: a practitioner's approach*. Palgrave Macmillan.
[13] Michael Rath and Patrick Mäder. 2019. Replication Data for: Request for Comments: Conversation Patterns in Issue Tracking Systems of Open-Source Projects. https://bit.ly/38d5AYk. (2019). https://doi.org/10.7910/DVN/M8WTHU
[14] Michael Rath, Patrick Rempel, and Patrick Mäder. 2017. The IlmSeven Dataset. In *25th IEEE International Requirements Engineering Conference, RE 2017*.
[15] Patrick Rempel and Patrick Mäder. 2015. Estimating the Implementation Risk of Requirements in Agile Software Development Projects with Traceability Metrics. In *Requirements Engineering: Foundation for Software Quality - 21st Int. Working Conference, REFSQ*, Samuel A. Fricker and Kurt Schneider (Eds.). Springer.
[16] Patrick Rempel and Patrick Mäder. 2017. Preventing Defects: The Impact of Requirements Traceability Completeness on Software Quality. *IEEE Trans. Software Eng.* (2017). https://doi.org/10.1109/TSE.2016.2622264
[17] Peter C. Rigby and Ahmed E. Hassan. 2007. What Can OSS Mailing Lists Tell Us? A Preliminary Psychometric Text Analysis of the Apache Developer Mailing List. In *Fourth International Workshop on Mining Software Repositories, MSR*. IEEE Computer Society. https://doi.org/10.1109/MSR.2007.35
[18] Emad Shihab, Zhen Ming Jiang, and Ahmed E. Hassan. 2009. Studying the use of developer IRC meetings in open source projects. In *25th IEEE International Conference on Software Maintenance ICSM*. IEEE Computer Society.
[19] Ian Skerrett and The Eclipse Foundation. 2011. *The Eclipse Community Survey 2011*. Technical Report.
[20] Margaret-Anne D. Storey and Alexey Zagalsky. 2016. Disrupting developer productivity one bot at a time. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE*. ACM.
[21] Shaowei Wang and David Lo. 2014. Version history, similar report, and structure: putting them together for improved bug localization. In *22nd International Conference on Program Comprehension, ICPC*. ACM.
[22] Jian Zhou, Hongyu Zhang, and David Lo. 2012. Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports. In *34th International Conference on Software Engineering, ICSE*.