A Mixed Graph-Relational Dataset of Socio-technical Interactions in Open Source Systems

Usman Ashraf
Christoph Mayr-Dorn
Alexander Egyed
Johannes Kepler University, Austria
firstname.lastname@jku.at

Sebastiano Panichella
Zurich University of Applied Sciences, Switzerland
sebastiano.panichella@zhaw.ch

ABSTRACT

Several researchers have studied that developers contributing to open source systems tend to self-organize in “emerging” teams. The structure of these latent teams has a significant impact on software quality, with development teams structure somewhat reflected in the way developers communicate and contribute in the subsystems of a system. Therefore, in order to study socio-technical interactions as well as the software evolution dynamics of open source systems, in this paper, we present a novel dataset, gathered from 20 open source projects, which report the developers’ activities in the scope of commits and issues at the level of subsystems. Thus, the new, generated dataset comprises of emerging and explicit links among developers, commits, issues, and source code artifacts, with data grouped around the subsystems point of view, which can be used to better study the system dynamics behind the extracted socio-technical interactions.

CCS CONCEPTS

• **Software and its engineering → Collaboration in software development**.

KEYWORDS

socio-technical congruence, developer interactions, modularity

ACM Reference Format:
Usman Ashraf, Christoph Mayr-Dorn, Alexander Egyed, and Sebastiano Panichella. 2018. A Mixed Graph-Relational Dataset of Socio-technical Interactions in Open Source Systems. In *Woodstock ’18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

In industrial organizations most of the team building decisions are influenced by the project domain and organization size [1]. Vice versa, in open source system developers tend to spread across the world, working in different time zones [2, 3] and contributing to their desired part of the system i.e., subsystems. Most of developers’ socio-technical interactions happens using electronic means (e.g., mailing lists, issue tracking systems, IRC chats) [4, 5].

Many researchers studied different aspects of socio-technical interactions of open source systems [6–9]. We argue that, in the case of contemporary systems— e.g., systems based on microservices architectures and DevOps infrastructures— it’s likely that developers contributing to such systems often self-organize in “emerging” teams [10, 11]. Which are somewhat reflected in the way developers communicate and contribute in the subsystems of a system. Therefore, in order to study socio-technical interactions of open source systems, we could observe developers’ involvement in commits and issues at the level of subsystems.

Research Opportunities We prepared a dataset based on developers, commits, issues, source code artifacts, and the emerging and explicit links among these entities. We link these entities to their corresponding subsystem. Our dataset allows researchers to investigate how developers interact, through issues fixing activities [12] or (temporal and) similar (or conceptual related) commits with other contributors, also determining other *“socio-technical roles”* [4, 5] played by developers working in various subsystems.

Related Available Datasets Rath et al. [13] provide a smaller dataset, built from seven open source software projects, which enable to study social aspects of developers. However, developers in their dataset are not de-duplicated (i.e., developers do not use different accounts) and commits are not linked to corresponding subsystems. Moreover, less source of information have been considered to study developer interactions as issue comments are not a part of their dataset.

2 DATASET OVERVIEW

Our dataset consists of socio-technical information extracted from 20 projects hosted on Github (limited in number and size due to the manual processing effort required in step 1 and step 5 (Fig.1)). Out of these, 8 projects use the Apache Jira server for managing issues (aside from GitHub for managing pull requests) and the remainder use Github Issues exclusively. The dataset contains unique artifacts across commits, with unique issues and developers involvement such as code committing, issue commenting, and issue watching. Moreover, artifacts and issues have explicit links to subsystems. This dataset is provided as a MySQL database which can serve as a base to extract socio-technical information. Moreover, our dataset includes two Neo4J[1] graphs for each project based on developer and subsystem interactions respectively.

[1]https://neo4j.com/

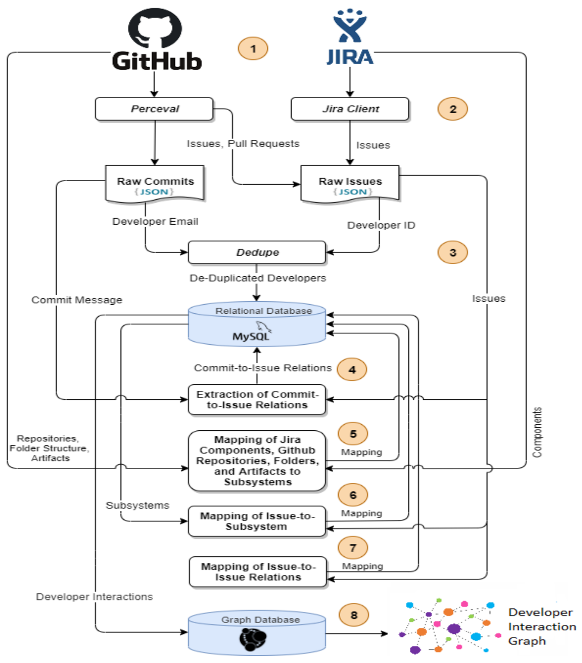
GitHub
JIRA
1
Perceval
Jira Client
2
Issues, Pull Requests
Issues
Raw Commits (JSON)
Raw Issues (JSON)
Developer Email
Developer ID
Dedupe
3
De-Duplicated Developers
Commit Message
Issues
Relational Database
MySQL
Commit-to-Issue Relations
4
Extraction of Commit-to-Issue Relations
Repositories, Folder Structure, Artifacts
5
Mapping of Jira Components, Github Repositories, Folders, and Artifacts to Subsystems
Mapping
Components
Subsystems
6
Mapping of Issue-to-Subsystem
Mapping
7
Mapping of Issue-to-Issue Relations
Mapping
Developer Interactions
Graph Database
8
Developer Interaction Graph

Figure 1: Steps of the dataset construction

3 DATASET CONSTRUCTION

3.1 Project Selection

Candidate projects in our study (Fig.1(Step 1)) are selected according to the following requirements:

1. Subsystem Structure Projects needed to exhibit a non-trivial (i.e., at least 10 subsystems) structure that reflects the real underlying decomposition. For micro-services based projects, we additionally aimed to select projects that manage code across multiple repositories (multi-repo), i.e., one repository per subsystem, as well as in a single repository (mono-repo), i.e., one folder per subsystem. For selecting monolithic projects having an explicit set of subsystems, we manually inspected all projects on the Apache Jira server (below 10000 issues due to manual processing effort) that exhibited a significant number of issues with a *component* property set and where the top level source code folders (hosted on Github) closely (or identically) matched those component names. We interpreted those components/folders as subsystems. For the purpose of selecting microservices-based projects, we searched on Github with queries consisting of specific terms such as *Microservices Architecture*, *Spring Cloud*, *Netflix OSS*, and *Kubernetes*. We manually classified the projects into mono-repo or multi-repo. For mono-repo micro-services projects, we manually inspected the folder names for clear indication that a folder represents one subsystem.

2. Active developers We selected projects with at least 40 contributing developers over the project's life time. This guarantees that there are sufficiently many developers involved to study meaningful socio-technical interaction patterns.

3. Commits and issues Furthermore, we filtered out projects with less than 1,500 commits or 200 issues. This ensures that a relation between two developers based on editing the same artifact or being active in the same issue may occur sufficiently often to signify a meaningful relation between the developers.

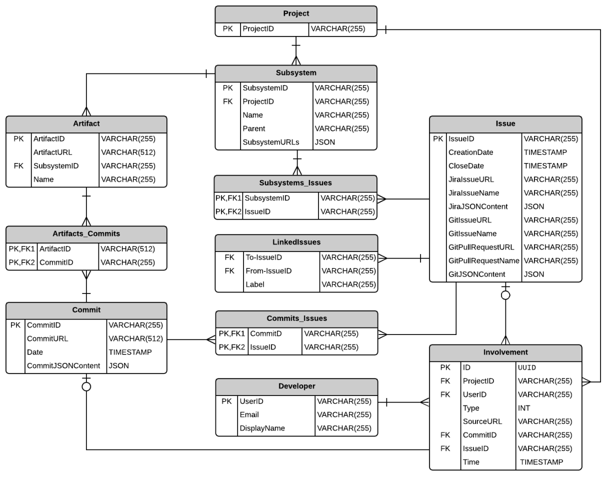

Figure 2: Data Schema - Relational Part

3.2 Data Extraction and Storing

We used *Perceval* [14] to extract commits and issues information from Github (Step 2). For the purpose of extracting data from Apache Jira we used the Jira python client.

Figure 2 depicts a simplified relational data model overview. *Commits*, the involved *Artifacts*, and *Issue* details make up the core information of our data set. In our dataset an *Issue* may represent a Jira issue, a Github Issue, or a Github Pull request (or a combination, e.g., a pull request references a Jira issue).

A developer may contribute in a project in various ways: for example, committing code changes, commenting of issues, reviewing pull requests, or watching an issue. We introduce three *Involvement* types to harmonize activities across Jira and GitHub. *Contributing* is equal to committing artifacts; *Informative* describes input to an issue such as having reported it, commented on it, or having reviewed artifacts; *Interested* describes passive actions such as watching an issue or being assignee without having commented or committed yet. We encode these three *types* of actions as integers of value 10, 5, and 1, respectively, to reflect the amount of effort behind the activities.

3.3 Developer De-Duplication

When recording a developer's involvement across commits and issues, we need to take care of situations where a developer uses multiple accounts (email-ids) on Github. More importantly, a developer's id on Jira doesn't match the Github account id of the same developer as Jira doesn't use email addresses as part of a user id while Github does. This will result in inconsistencies as one developer will appear as multiple individual in the data set, hence the need for deduplication (Step 3).

We used the *Dedupe*[2] library which uses machine learning to perform deduplication. The input to *Dedupe* is a list of triples, each

[2]https://github.com/dedupeio/dedupe

containing a *processed display name*, a *processed identifier*, and a *derived short identifier*.

Dedupe Training: We trained and evaluated *Dedupe* on a sample Jira-backed open source project achieving 100% precision and recall. We saved results of our training in a JSON file and ran *Dedupe* for the complete set of user details from all 20 projects. We store for each developer the list of duplicates, using the first encountered identifier (email in Github or Jira user id) as the UserID in the *Developer* table (Fig. 2) and use the duplicate list to ensure that all involvements in Jira and Github of the same developer (regardless of their used identity) point to the same developer in our data set.

3.4 Extraction of Commit-To-Issue Relations

Github commit messages often refer to issues to identify the purpose of the changes. We manually inspected each project what patterns developers tend to apply for identifying issues and applied that pattern when parsing the commit messages (Step 4).

We apply the same mechanism to pull requests. Whenever a pull request identifies a Github issue, we merge pull request and Github issue into one data set *Issue* and link all commits that are part of the pull request to it. This is particularly helpful when commit messages lack a reference to an issue.

For the Jira-backed projects we regularly find pull requests and commits without a reference to a Github issue but instead a reference to a Jira issue. Whenever a pull request identifies a Jira issue, we subsequently merged them into a single *Issue* in our data set (we never encountered multiple pull requests referencing the same issue) and link all involvements from the pull request and Jira issue to that merged Issue. We proceed likewise whenever we find a Github issue referencing (in the comments or title) a Jira issue.

3.5 Subsystem Mapping

For those projects that utilize Apache Jira for issue management we need to map a Jira project's components to the corresponding folders on Github (Step 5). We manually mapped each component of the eight Jira projects to their corresponding Github folders (multiple folders where necessary to map all Github folders).

The mapping for Github multi-repo projects is straight forward: each repository in the project becomes a subsystem. In the case of a Github mono-repo (non-Jira backed), we determined the subsystem based on manual inspection of folder names.

In our datamodel the *Subsystem* table's *SubsystemURLs* JSON blob contains the list of folders that make up a subsystem. This allows us to relate all *Artifacts* to a unique *Subsystem* by matching a code artifact's URL path to the subsystem's URL.

3.6 Linking Issues to Subsystems

Linking *Issues* to *Subsystems* (Step 6) works similar relating *Artifacts* to *Subsystems*. Jira issues exhibit a *Component/s* property that identifies all affected subsystems (potentially multiple). Github multi-repo projects provide separate issue lists for each repository, thus the link to subsystems is unambiguous there as well. For the two Github mono-repo projects, we need to identify subsystems by traversing from *Issues* via *Commits* and *Artifacts* to *Subsystems*. Hence, here, we can only link those issues to subsystems that are also linked to commits.

3.7 Linking Issues to Issues

Issues may point to other issues from explicit fields (only available in Jira). For all such Jira-backed projects we store these links (Step 7) in *LinkedIssues* table.

4 DATASET STATISTICS

Table 1 provides an overview of the 20 chosen open source projects. *Type* column specifies project structure and *Time Period* column describes the overall time (in months) in which we extracted commits and issues. All subsequent rows report values from this time period. The *Commits* column reports how many commits are made in total in a project.[3] Recall that in multi-repo projects, commits per subsystem belong to one repository, thus the reported numbers are the sum of commits over all repositories in such a project. In case of mono-repo projects, all commits come from a single, main repository. The *Dev* column provides the total number of developers in each projects. We consider any person a developer who is *contributing*, *informative*, or *interested* in at least one subsystem. *Subsys*, *Art*, and *Issue* columns report the number of subsystems, total source code artifacts, and issues in a project, respectively. The columns *Arts Linked to Subsys* and *Issues Linked to Subsys* show the percentage of artifacts and issues linked to subsystems respectively. Similarly, the columns *Issue Linked to Commits*, and *Commits Linked to Issue* provide the percentage of issues linked to commits and vice versa, respectively. Finally, the column *Dev Interaction Graph Density* represents the density value of each developer interaction graph (see section 5.1).

5 DATASET: GRAPH-BASED PART

We use graphs to give an interactive and easy representation of socio-technical information. The MySQL database provides the base data and context for the extraction of information which we export to a graph database (Neo4J). The following section describes the graphs we include in our dataset.

5.1 Developers Interaction Graph

We built developer interaction graph from developers' involvement in issues and editing of source code artifacts over the life time of the project (Fig. 1 (8)). We model the developer interaction graph with developers as nodes ($d \in V$) and edges ($e(d_i, d_k) \in E$) between those developers that interact. Specifically we add an edge between two developers when at least one of the following two conditions are fulfilled: First, the two developers changed at least once the same file within the temporal proximity of 4 months. The edge's *common changed artifact count* property stores how many distinct source code artifacts were changed by both developers. Second, the two developers are involved in the same issue, for example, one developer adding a comment to an issue, the other developer committing an artifact update linked to the same issue. The edge's *issue involvement intensity* measures in how many issues the two developers are active in, weighted by their involvement type.

The linear combination of *common changed artifact count* and *issue involvement intensity* produces a single edge weight that balances contributing source code changes with issue management.

[3]Note that GitHub only shows the number of commits from master branch. However, *Perceval* fetches commits from all branches in the repository including master branch.

Table 1: Overview of the analysed projects.

Project	Type	Time Period [Months]	Commits	Dev	Subsys	Arts	Issues	Arts Linked to Subsys [%]	Issues Linked To Subsys [%]	Issues Linked To Commits [%]	Commits Linked To Issues [%]	Dev Interaction Graph Density
lagom	Multi-Repo	44	6089	540	17	27490	3381	100	100	59	62	0.05
nameko	Multi-Repo	85	3861	233	14	960	867	100	100	65	93	0.02
kumuluz	Multi-Repo	54	2487	83	19	2083	274	100	100	45	22	0.06
jhipster	Multi-Repo	71	10391	767	11	7146	3267	100	100	64	59	0.01
networknt	Multi-Repo	38	5369	110	12	16127	2005	100	100	74	36	0.04
ibm-cloud-architecture	Multi-Repo	30	3519	54	15	2207	593	100	100	72	70	0.08
MitocGroup	Multi-Repo	38	5513	46	16	8270	1825	100	100	65	86	0.17
nodezoo	Multi-Repo	62	1589	65	13	1070	687	100	100	59	62	0.11
spring-petclinic	Multi-Repo	74	1579	148	11	4603	261	100	100	51	19	0.02
microservices-demo	Multi-Repo	36	3122	139	20	2297	1314	100	100	56	81	0.05
flume	Mono-Repo (Jira)	114	2623	1199	16	3403	3638	51	57	29	86	0.01
stanbol	Mono-Repo (Jira)	107	6953	149	21	17469	1490	54	80	47	62	0.04
genie	Mono-Repo	68	2905	71	16	3873	829	54	0	32	37	0.07
spring-cloud-dataflow	Mono-Repo	65	2129	247	17	2383	2991	62	0	29	67	0.04
batik	Mono-Repo (Jira)	221	3711	755	18	8760	1274	52	91	4	4	0.01
falcon	Mono-Repo (Jira)	87	2556	190	19	7362	2760	56	38	47	63	0.07
arrow	Mono-Repo (Jira)	47	3772	942	24	4431	8717	96	48	36	95	0.03
tika	Mono-Repo (Jira)	154	5848	1327	21	6509	3335	71	65	46	67	0.01
openjpa	Mono-Repo (Jira)	164	7215	831	27	10696	2849	54	73	57	68	0.01
oodt	Mono-Repo (Jira)	118	2772	103	23	38200	1138	60	78	20	31	0.07

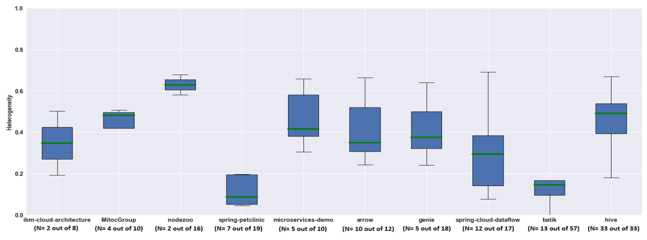

Figure 3: Range of average subsystem developer team membership heterogeneity per project.

5.2 Subsystem Interaction Graph

We provide a subsystem interaction graph as a counterpart to the developer interaction graph. Here nodes ($sy \in V$) are subsystems and edges ($e(sy_i, sy_k) \in E$) between them describe dependencies based on i) links among issues that belong to different subsystems and ii) and developers that are involved in issues that belong to different subsystems. In the former case, the edge identifies all issues (*issue count*) with cross-subsystem references. In the latter case, the edge identifies the developers (*developer count*) that are informative or contributing in the two linked subsystems. We combine the *issue count* and the *developer count* to produce an edge's weight.

6 DATA USAGE

6.1 Using the dataset

Our supporting online material (SOM) [4] includes relational dataset, per-project interaction graphs, and source code. The relational dataset (*Dump.sql*) includes the schema (see Fig. 2) for importing to a local MySQL server. The *graphs.cypher* has the *cypher* [5] script for developer and subsystem interaction graphs of each project which can be created and viewed using Neo4j browser.[6] The *src* contains the Python code for extraction of raw data, preprocessing, and interaction graph calculation.

[4]http://doi.org/10.5281/zenodo.3707756
[5]https://neo4j.com/developer/cypher-query-language/
[6]https://neo4j.com/developer/neo4j-browser/

6.2 Example Research Questions

We analyzed projects from our dataset to detect latent developer communities and investigate the overlap (Heterogeneity) of the subsystem developer teams with these detected communities. This allows us to identify subsystems where the communication among developers is non-optimal. The figure 3 shows our results.

Our dataset can be valuable in order to find the answers of research questions e.g., *"How does socio-technical congruence manifest in micro-services architecture? How do developer networks span across subsystems? What is the impact of socio-technical interaction patterns in the number and average time to close issue reports?"*. Moreover, for studying potential differences between mono-repo and multi-repo projects, respectively monolithic and micro-services based systems.

7 CHALLENGES AND LIMITATIONS

We cannot be absolutely certain that the mapping between Jira issue components and GitHub folders is absolutely correct as a non negligible manual effort is necessary to investigate folder content.

We did not consider other possible developer communication channels i.e., mailing lists and IRC etc. Panichella et al. [4] found that communication links obtained from mailing lists have high overlap with links obtained from issue trackers and that mining links from chat is less reliable as this (i) tends to produce too many links and (ii) conversations are less easily associated to issues. We, therefore, believe that we miss only negligible information by leaving out mailing list and chats.

8 CONCLUSION

In this paper we described and published our mixed relational and graph based dataset which is based on developers' activities from 20 projects. These activities are in the scope of commits and issues and grouped to subsystems. We provided details regarding dataset construction and its usage. We hope that our dataset will help the research community to study novel aspects of socio-technical interactions.

ACKNOWLEDGMENTS

This work was supported by the Austrian Science Fund (FWF): P29415-NBL funded by the Government of Upper Austria.

REFERENCES

[1] Jorge Aranda and Gina Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. In *International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*, pages 298–308, 2009.

[2] Eugenio Capra and Anthony I. Wasserman. A framework for evaluating managerial styles in open source projects. In *Open Source Development, Communities and Quality, IFIP 20th World Computer Congress, Working Group 2.3 on Open Source Software, OSS 2008, September 7-10, 2008, Milano, Italy*, pages 1–14. Springer, 2008.

[3] Andrew Begel and Nachiappan Nagappan. Global software development: Who does it? In *3rd IEEE International Conference on Global Software Engineering, ICGSE 2008*, pages 195–199, 2008.

[4] Sebastiano Panichella, Gabriele Bavota, Massimiliano Di Penta, Gerardo Canfora, and Giuliano Antoniol. How developers' collaborations identified from different sources tell us about code changes. In *International Conference on Software Maintenance and Evolution, 2014*, pages 251–260.

[5] Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. Who is going to mentor newcomers in open source projects? In *Proceedings of the 20th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, page 44, Cary, NC, USA, 2012.

[6] Mohammad Gharehyazie and Vladimir Filkov. Tracing distributed collaborative development in apache software foundation projects. *Empirical Software Engineering*, 22(4):1795–1830, 2017.

[7] Bora Çaglayan and Ayse Basar Bener. Effect of developer collaboration activity on software quality in two large scale projects. *Journal of Systems and Software*, 118:288–296, 2016.

[8] Jing Wu, Khim-Yong Goh, He Li, Chuan Luo, and Haichao Zheng. The effects of communication patterns on the success of open source software projects: An empirical analysis from social network perspectives. *JGIM*, 24(4):22–44, 2016.

[9] Daniela E. Damian, Remko Helms, Irwin Kwan, Sabrina Marczak, and Benjamin Koelewijn. The role of domain knowledge and cross-functional communication in socio-technical coordination. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 442–451, 2013.

[10] Valentina Lenarduzzi and Outi Sievi-Korte. On the negative impact of team independence in microservices software development. In *Proceedings of the 19th International Conference on Agile Software Development: Companion*, XP '18, New York, NY, USA, 2018. Association for Computing Machinery.

[11] Christian Bird, David Pattison, Raissa D'Souza, Vladimir Filkov, and Premkumar Devanbu. Latent social structure in open source projects. In *ACM SIGSOFT International Symposium on Foundations of software engineering*, SIGSOFT '08/FSE-16, pages 24–35. ACM, 2008.

[12] Gerardo Canfora, Luigi Cerulo, Marta Cimitile, and Massimiliano Di Penta. Social interactions around cross-system bug fixings: the case of freebsd and openbsd. In Arie van Deursen, Tao Xie, and Thomas Zimmermann, editors, *International Working Conference on Mining Software Repositories, MSR 2011 (Co-located with ICSE)*, pages 143–152. ACM, 2011.

[13] Michael Rath, Patrick Rempel, and Patrick Mäder. The ilmseven dataset. In *25th IEEE International Requirements Engineering Conference, RE 2017, Lisbon, Portugal, September 4-8, 2017*, pages 516–519, 2017.

[14] Santiago Dueñas, Valerio Cosentino, Gregorio Robles, and Jesus M Gonzalez-Barahona. Perceval: software project data at your will. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pages 1–4. ACM, 2018.