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:

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 graphs for each project based on developer and subsystem interactions respectively.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Woodstock ’18, June 03–05, 2018, Woodstock, NY © 2018 Association for Computing Machinery. ACM ISBN 978-1-4503-XXXX-X/18/06 ... $15.00 https://doi.org/10.1145/1122445.1122456

1https://neo4j.com/
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.

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 library which uses machine learning to perform deduplication. The input to Dedupe is a list of triples, each

2https://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 straightforward: 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. 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 project. 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 (\(e(d_i, d_j) \in E\)) 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.

---

3Note that Github only shows the number of commits from master branch. However, Perceval fetches commits from all branches in the repository including master branch.
We provide a subsystem interaction graph as a counterpart to the
which can be created and viewed using Neo4j browser. 

Our supporting online material (SOM)
includes the schema (see Fig. 2) for importing
or contributing in the two linked subsystems. We combine the
based on i) links among issues that belong to
different subsystems. In the former case, the edge identifies all issues
with links obtained from issue trackers and that mining links from
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.

5.2 Subsystem Interaction Graph
We provide a subsystem interaction graph as a counterpart to the
developer interaction graph. Here nodes (\(s_y \in V\)) are subsystems
and edges (\(e(s_y, s_y) \in E\)) between them describe dependencies based on
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
... we describe and publish 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.

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.

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.

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

Table 1: Overview of the analysed projects.

<table>
<thead>
<tr>
<th>Project</th>
<th>Type</th>
<th>Time Period [Months]</th>
<th>Commits</th>
<th>Dev</th>
<th>Subsys</th>
<th>Arts</th>
<th>Issues</th>
<th>Arts Linked To Subsys [%]</th>
<th>Issues Linked To Subsys [%]</th>
<th>Issues Linked To Commits [%]</th>
<th>Issues Linked To Issues [%]</th>
<th>Commits Linked To Issues [%]</th>
<th>Dev Interaction Graph Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>lagom Multi-Repo</td>
<td></td>
<td>4</td>
<td>6089</td>
<td>398</td>
<td>4500</td>
<td>935</td>
<td>5384</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0.05</td>
</tr>
<tr>
<td>nameko Multi-Repo</td>
<td></td>
<td>1</td>
<td>386</td>
<td>38</td>
<td>14</td>
<td>39</td>
<td>34</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0.02</td>
</tr>
<tr>
<td>kumiko Multi-Repo</td>
<td></td>
<td>4</td>
<td>2158</td>
<td>283</td>
<td>1400</td>
<td>704</td>
<td>566</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0.03</td>
</tr>
<tr>
<td>jhipster Multi-Repo</td>
<td></td>
<td>1</td>
<td>203</td>
<td>38</td>
<td>12</td>
<td>32</td>
<td>28</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0.06</td>
</tr>
<tr>
<td>networkkit Multi-Repo</td>
<td></td>
<td>2</td>
<td>150</td>
<td>38</td>
<td>15</td>
<td>26</td>
<td>19</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0.06</td>
</tr>
<tr>
<td>hm-cloud-architecture  Multi-Repo</td>
<td></td>
<td>2</td>
<td>150</td>
<td>38</td>
<td>15</td>
<td>26</td>
<td>19</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0.06</td>
</tr>
<tr>
<td>MitoGroup Multi-Repo</td>
<td></td>
<td>3</td>
<td>551</td>
<td>46</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0.01</td>
</tr>
<tr>
<td>nodeflow Multi-Repo</td>
<td></td>
<td>3</td>
<td>551</td>
<td>46</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0.01</td>
</tr>
<tr>
<td>spring-petrikin Multi-Repo</td>
<td></td>
<td>5</td>
<td>1579</td>
<td>148</td>
<td>11</td>
<td>46</td>
<td>36</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0.02</td>
</tr>
<tr>
<td>microservices-demo     Multi-Repo</td>
<td></td>
<td>3</td>
<td>3122</td>
<td>159</td>
<td>20</td>
<td>229</td>
<td>1314</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0.05</td>
</tr>
<tr>
<td>flume Mono-Repo (Jira)</td>
<td></td>
<td>114</td>
<td>2823</td>
<td>1399</td>
<td>16</td>
<td>3403</td>
<td>3638</td>
<td>51</td>
<td>57</td>
<td>29</td>
<td>86</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>stanbol Mono-Repo (Jira)</td>
<td></td>
<td>107</td>
<td>6953</td>
<td>149</td>
<td>21</td>
<td>17469</td>
<td>14900</td>
<td>54</td>
<td>80</td>
<td>47</td>
<td>62</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>gse Mono-Repo</td>
<td></td>
<td>68</td>
<td>2905</td>
<td>71</td>
<td>16</td>
<td>3873</td>
<td>829</td>
<td>54</td>
<td>0</td>
<td>32</td>
<td>37</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>spring-cloud-dataflow  Mono-Repo</td>
<td></td>
<td>65</td>
<td>2129</td>
<td>247</td>
<td>17</td>
<td>2583</td>
<td>2991</td>
<td>62</td>
<td>0</td>
<td>29</td>
<td>67</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>bitk Mono-Repo (Jira)</td>
<td></td>
<td>221</td>
<td>3711</td>
<td>755</td>
<td>18</td>
<td>8760</td>
<td>1274</td>
<td>52</td>
<td>91</td>
<td>4</td>
<td>4</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>falcon Mono-Repo (Jira)</td>
<td></td>
<td>87</td>
<td>2556</td>
<td>190</td>
<td>19</td>
<td>3762</td>
<td>2760</td>
<td>56</td>
<td>38</td>
<td>47</td>
<td>63</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>arrow Mono-Repo (Jira)</td>
<td></td>
<td>47</td>
<td>3772</td>
<td>942</td>
<td>24</td>
<td>4431</td>
<td>8717</td>
<td>96</td>
<td>48</td>
<td>36</td>
<td>95</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>tika Mono-Repo (Jira)</td>
<td></td>
<td>154</td>
<td>5468</td>
<td>1327</td>
<td>21</td>
<td>6509</td>
<td>3335</td>
<td>71</td>
<td>65</td>
<td>46</td>
<td>67</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>opesys Mono-Repo (Jira)</td>
<td></td>
<td>164</td>
<td>7215</td>
<td>831</td>
<td>27</td>
<td>16036</td>
<td>2849</td>
<td>54</td>
<td>73</td>
<td>57</td>
<td>64</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>osdl Mono-Repo (Jira)</td>
<td></td>
<td>118</td>
<td>2772</td>
<td>103</td>
<td>23</td>
<td>38280</td>
<td>1138</td>
<td>40</td>
<td>78</td>
<td>20</td>
<td>31</td>
<td>0.07</td>
<td></td>
</tr>
</tbody>
</table>

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

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


