An Exploratory Study of Socio-Technical Congruence in an Ecosystem of Software Developers

by

Deepak Azad

Bachelor of Engineering, University of Delhi, India, 2007

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Computer Science)

The University of British Columbia
(Vancouver)

December 2014

© Deepak Azad, 2014
Abstract

Software is not built in isolation but builds on other software. When one project relies on software produced by another project, we say there is a technical dependence between the projects. The socio-technical congruence literature suggests that when there is a technical dependence there may need to be a social dependence. We investigate the alignment between social interactions and technical dependence in a software ecosystem.

We performed an exploratory study of 250 Java projects on GitHub that use Maven for build dependences. We create a social interaction graph based on developers’ interactions on issue and pull requests. We compare the social interaction graph with a technical dependence graph representing library dependences between the projects in the ecosystem, to get an overview of the congruence, or lack thereof, between social interactions and technical dependences. We found that in 23.6% of the cases in which there is a technical dependence between projects there is also evidence of social interaction between project members. We found that in 8.67% of the cases in which there is a social interaction between project members, there is a technical dependence between projects.

To better understand the situations in which there is congruence between the social and technical graphs, we examine pairs of projects that meet this criteria. We identify three categories of these project pairs and provide a quantitative and qualitative comparison of project pairs from each category. We found that for 45 (32%) of project pairs, no social interaction had taken place before the introduction of technical dependence and interactions after the introduction of the dependence are often about upgrading the library being depended upon. For 49 (35%) of project pairs, 75% of the interaction takes place after the introduction of the technical...
dependence. For the remaining 45 (32%) of project pairs, less than 75% of the interaction takes place after the introduction of the technical dependence. In the latter two cases, although there is interaction before the technical dependence is introduced, it is not always about the dependence.
Preface

The work presented in this thesis is original, unpublished, independent work by the author and was conducted in the Software Practices Lab under supervision of Prof. Gail Murphy.
# Table of Contents

Abstract ................................................................. ii
Preface ................................................................. iv
Table of Contents ....................................................... v
List of Tables .......................................................... vii
List of Figures .......................................................... viii
Acknowledgments ....................................................... xi
Dedication ............................................................... xii

1 Introduction ......................................................... 1

2 Related Work ......................................................... 3
   2.1 Socio-Technical Congruence .................................... 3
   2.2 Developer Social Networks ..................................... 5

3 Dataset Description ................................................ 6

4 Constructing the Graphs .......................................... 9
   4.1 Technical Dependence Graph .................................. 9
   4.2 Social Project Graph ........................................... 10
      4.2.1 User Interaction Graph ................................... 11
      4.2.2 Forming the Social Project Graph ....................... 13
List of Tables

Table 3.1 A characterization of the 250 projects studied between 1 January 2010 and 2 April 2014 ......................... 6

Table 5.1 Overview of TDG and SPG ............................... 20
List of Figures

Figure 4.1  An example technical dependence graph with four projects p, q, m, n. Each edge is annotated with the date the technical dependence was introduced and the technical dependence at the time. ......................................................... 10

Figure 4.2  An example user interaction graph with only two users x, y. These users interact on three projects p, q, n. Each edge is annotated with the project to which the corresponding user interactions belong, and each edge is weighted by the number of times the two users interacted with each other on that project. . 12

Figure 4.3  Degree of nodes in UIG  ......................................................... 13

Figure 4.4  An example social project graph with four projects p, q, m, n. Each edge is annotated with all the actions of the users on the two projects who were active on both the projects. .......... 14

Figure 4.5  Community detection in user interaction graph  .......... 15

Figure 4.6  Communities detected  ......................................................... 16

Figure 4.7  Example communities detected in UIG. Nodes are users and the edges are the interaction between users. Different edge colors within a community represent the different projects over which the interactions occurred. ................................. 17

Figure 5.1  Degree of nodes in TDG and SPG  ......................... 21

Figure 5.2  Percentage activity after technical dependency is introduced  . 22

Figure 5.3  Category A - wildfly/wildfly(blue) and resteasy/Resteasy(green)  ......................................................... 23
Figure 5.4  Category B - infinispan/infinispan(blue) and weld/core(green) . 25
Figure 5.5  Category C-sarxos/webcam-capture(blue) and netty/netty(green) .................................................. 26

Figure 6.1  Distances between users in a community .......... 31

Figure C.1  Category A project pairs (1-6) ................. 53
Figure C.2  Category A project pairs (7-12) ............ 54
Figure C.3  Category A project pairs (13-18) ......... 55
Figure C.4  Category A project pairs (19-24) ......... 56
Figure C.5  Category A project pairs (25-30) ......... 57
Figure C.6  Category A project pairs (31-36) ......... 58
Figure C.7  Category A project pairs (37-42) ......... 59
Figure C.8  Category A project pairs (43-45) ......... 60

Figure D.1  Category B project pairs (1-6) ............ 62
Figure D.2  Category B project pairs (7-12) ............ 63
Figure D.3  Category B project pairs (13-18) ......... 64
Figure D.4  Category B project pairs (19-24) ......... 65
Figure D.5  Category B project pairs (25-30) ......... 66
Figure D.6  Category B project pairs (31-36) ......... 67
Figure D.7  Category B project pairs (37-42) ......... 68
Figure D.8  Category B project pairs (43-48) ......... 69
Figure D.9  Category B project pair (49) ............... 70

Figure E.1  Category C project pairs (1-6) ............... 72
Figure E.2  Category C project pairs (7-12) ............. 73
Figure E.3  Category C project pairs (13-18) ......... 74
Figure E.4  Category C project pairs (19-24) ......... 75
Figure E.5  Category C project pairs (25-30) ......... 76
Figure E.6  Category C project pairs (31-36) ......... 77
Figure E.7  Category C project pairs (37-42) ......... 78
Figure E.8  Category C project pairs (43-45) ......... 79

Figure F.1  Communities detected in UIG (1-6) ........... 81
Figure F.2  Communities detected in UIG (7-12) . . . . . . . . . . . . . . . 82
Figure F.3  Communities detected in UIG (13-18) . . . . . . . . . . . . . . . 83
Figure F.4  Communities detected in UIG (19-22) . . . . . . . . . . . . . . . 84
Acknowledgments

I am grateful to my supervisor Gail Murphy for her encouragement and guidance in pursuing an interesting topic for my Masters Thesis.

I thank Marc Palyart for his guidance in solidifying the research methodology. I also thank my second reader, Ivan Beschastnikh, who provided valuable feedback on my thesis draft.

Many thanks to my colleagues of Software Practices Lab who created a cheerful atmosphere in the lab and provided guidance when needed.

Finally, I would like to acknowledge NSERC for funding this research.
Dedication

To my amazing parents...
...for their endless love and support.
Chapter 1

Introduction

Software is not typically built in isolation, but instead builds on other software. When one project relies on software produced by another project, we say that there is a technical dependence between the projects. The socio-technical congruence literature suggests that when there is a technical dependence between the projects, there may need to be some social interactions between members of those projects [5, 6, 15]. To date, socio-technical congruence has been considered in the context of a single project or system (e.g., [5–7]), in a set of highly related projects that result in a simultaneous release (e.g., [19]) or over the lifetime of one system (e.g., [5, 7]). Socio-technical congruence has not been studied in the context of an ecosystem of software developers, in which a variety of projects may be developed and evolved largely independently but may share developers through use of a common infrastructure.

In this thesis, we investigate socio-technical congruence in the context of an ecosystem of software developers. We study socio-technical congruence across 250 projects that use Java and Maven\(^1\) on GitHub. We chose GitHub as an example of a software ecosystem because it hosts a large number of diverse open source software projects, includes independent projects for which there are technical interactions between the projects, and includes data on social interactions, such as comments on issues and pull requests. We form a technical dependence graph for the 250 projects based on Maven project information that details library depen-

\(^1\)http://maven.apache.org/
dences between projects. We form a social dependence graph by forming a social interaction graph based on communities detected from developer interactions on the various projects.

We investigate two research questions:

1. How often does congruence between technical dependencies and social interactions occur in an ecosystem of software developers?

2. When socio-technical congruence does exist, how does it come about?

We found that for 23.6% of the cases in which there is a technical dependence between two projects, there is also evidence of some social interactions. In only 8.67% of the cases when there is a social interaction between projects is there also a technical dependence. We categorized all cases in which there are both social and technical dependences between projects, finding that for 45 (32%) of these cases, no social interaction had taken place before the introduction of technical dependence. In another 49 (35%) of the cases, most of the social interaction (more than 75%) took place after the technical dependence was introduced. In the remaining 45 (32%) of the cases a small percentage of social interaction (less than 75%) took place after the introduction of technical dependence. We also qualitatively examined an instance from each category to provide insight into the different cases that arise.

We begin with a review of related work in socio-technical congruency and determination of developer social networks (Chapter 2). We then describe the GitHub data used in this investigation (Chapter 3) before detailing the construction of the technical and social graphs (Chapter 4). We present the results of our investigations (Chapter 5), discuss the approach and results (Chapter 6) and summarize (Chapter 7).
Chapter 2

Related Work

Coordination among project members has been recognized as one of the fundamental problems in software engineering (e.g., [9, 12, 14]). It can affect many development qualities including developer productivity [6], build success probability [15] and software failures [5]. We describe previous efforts in socio-technical congruence and approaches for analyzing social networks amongst developers.

2.1 Socio-Technical Congruence

Conway first proposed the idea that structure of a system usually resembles the communication structure of the organization that designs it [8]. This concept has been explored in engineering [4] and management science [18]. In software engineering Cataldo et al. focused attention on socio-technical congruence [6]. They defined socio-technical congruence as the match between the coordination needs established by the technical dimension of the socio-technical system and the actual coordination activities carried out by the project members representing the social dimension.

Previous work has looked at congruence in a single product or system or software that is released together. Cataldo et al. investigated congruence in a single large distributed system developed by one company involving 114 developers [6, 7]. Later, Cataldo and Herbsleb also investigated congruence in a single complex embedded system developed by a single organization involving 380 de-
Syeed and Hammouda empirically examined Conway’s law in the FreeBSD open source project that is developed by a team of individuals as opposed to a single organization [19]. FreeBSD involved 1128 contributors in the last release who developed 20 packages that were released together. In contrast, we explore congruence between technical dependencies and social interactions in an ecosystem of software developers who work on independent projects which are not necessarily released together.

High socio-technical congruence is often considered desirable. Cataldo et al. found that high Socio-Technical congruence is associated with increased developer productivity as measured in terms of time taken to resolve a change request[6]. Kwan et al. studied the relationship between Socio-Technical congruence and build success probability. They report that for continuous builds, increasing congruence improves the chance of build success, however increasing congruence can actually decrease build success probability in integration builds[15]. Cataldo and Herbsleb also studied the impact of Socio-Technical congruence on software failures, and found that lower congruence increased software failures[5].

Another direction of study has been the evolution of socio-technical congruence over the life cycle of a development project. Cataldo et al. examined the evolution of congruence across four releases on a project and found that congruence often improved or remained stable over time [7]. They also examined the evolution of congruence among developers who contributed the most and the rest of the developers. They found that congruence among developers who contributed the most increased more over time as compared to rest of the developers. This analysis was replicated on two different projects with similar results [5]. Syeed and Hammouda also examined the evolution of congruence value across several releases of FreeBSD open source project [19]. They found that the congruence value remains stable and increases gradually over time. However, little has been done to examine the origins of socio-technical congruence when it does exist.

The socio-technical congruence literature has considered a range of definitions for identifying coordination needs or technical dependencies. Cataldo et al. used dependencies among tasks [6, 7]. Syeed and Hammouda use dependencies in source code at a file level to establish coordination needs [19]. To enable proactive detection of coordination needs, Blincoe et al. use information about the activities
of developers associated with different tasks [2]. In this thesis, we use library dependencies as a natural unit for a technical dependence between two Java projects.

2.2 Developer Social Networks

Other researchers have focused on analysis of how developers interact separate from the technical structure of the system(s) they are developing. Bird et al. extracted developer social networks from mailing list archives of five large open source projects and identified the community structure from these social networks [1]. Hong et al. examined the community evolution patterns in developer networks based on Mozilla bug reports and observed the individual community evolution paths [13]. They also compare these developer social networks with general social networks. They find that the size of communities in developer social networks is small compared to that in most general social networks. They also find that developer social networks have a widespread community size distribution and their biggest community accounts for 21% to 36% of the total developers. Our formulation of the developer social network is similar to Hong et al. in that we use discussions on issues and pull requests as a means to identify developer interactions[13].
Chapter 3

Dataset Description

To explore the research questions of interest in this thesis, we needed a diverse set of projects through which many developers interact. We chose GitHub because it is a popular hosting service for open source software projects, and hosts a diverse set of projects. As the projects on GitHub are open-source, we can analyze the project source for technical dependencies. To make the determination of technical dependencies tractable, we focused on projects on GitHub in Java that use Maven.

Maven is a build automation tool primarily used for Java projects. Java projects that use Maven describe dependencies via Project Object Model (POM) \(^1\) files, which are XML files that contain information about the project, its configuration and all technical dependencies used by Maven to build the project. POM uniquely identifies a software artefact via three required fields - groupId, artifactId, version. GroupId is generally unique amongst an organization or a project. For example,

\(^1\)http://maven.apache.org/pom.html

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>Mean</th>
<th>Std. dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Issues</td>
<td>110,374</td>
<td>441.5</td>
<td>613.9</td>
</tr>
<tr>
<td>#Issue comments</td>
<td>236,240</td>
<td>945</td>
<td>2066.6</td>
</tr>
<tr>
<td>#Pull requests</td>
<td>61,370</td>
<td>254.6</td>
<td>488</td>
</tr>
<tr>
<td>#Pull request comments</td>
<td>45,902</td>
<td>257.9</td>
<td>510.8</td>
</tr>
</tbody>
</table>

Table 3.1: A characterization of the 250 projects studied between 1 January 2010 and 2 April 2014
all core Maven artifacts live under the *groupId* org.apache.maven. However, an organization may have several projects. The *artifactId* is generally the name that the project is known by. It, along with the *groupId*, creates a key that uniquely identifies a project. The *version* determines which incarnation of a project is being talked about. POM also includes a list of all the dependencies needed by a project for successful compilation. If a project uses Maven for its builds, all the dependencies for the project can be obtained from the POM.

Listing 3.1 provides an example of a POM file for the project cucumber-testng. The *groupId* for this project is derived from the *groupId* of its parent, info.cukes. This file also specifies two dependencies on two artefacts, cucumber-core which has the same *groupId*, and testng which is a project from another organization. In this way, we can analyze POM files to uniquely identify projects and extract their technical dependencies.

**Listing 3.1: Sample POM file**

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <parent>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-jvm</artifactId>
    <relativePath>../pom.xml</relativePath>
    <version>1.2.0-SNAPSHOT</version>
  </parent>
  <artifactId>cucumber-testng</artifactId>
  <packaging>jar</packaging>
  <name>Cucumber-JVM: TestNG</name>

  <dependencies>
    <dependency>
      <groupId>info.cukes</groupId>
      <artifactId>cucumber-core</artifactId>
    </dependency>
    <dependency>
      <groupId>org.testng</groupId>
      <artifactId>testng</artifactId>
    </dependency>
  </dependencies>
</project>
```
In this thesis we consider the top 250 Java projects that use Maven on GitHub based on the number of comments and activity on issues and pull requests. We consider the top 250 projects based on these criteria to ensure that we are analyzing projects with sufficient captured social interactions. We identify these projects using the GHTorrent dataset [11], and we use the GHTorrent MySQL dump from 2 April 2014.\(^2\)

Since Maven is used primarily with Java projects, we first identify Java projects. Github detects the primary language of each project and this is available in the GHTorrent data. We simply filter projects marked as 'Java' projects. Next we sort these projects based on number of comments on issues and pull requests. To filter Maven projects from the list of Java projects we detect presence of at least one POM file in a project repository. We obtain POM files by directly cloning the git repositories from GitHub and extracting the necessary files.

For these 250 projects, the total number of issues (110,374) is about twice the total number of pull requests (61,370). Also the total number of comments on issues (236,240) are about five times the total number of comments on pull requests (45,902). We consider the data from Jan 2010 onwards since the social interaction data is quite sparse before that. Table 3.1 summarizes these numbers about the dataset.

\(^2\)http://ghtorrent.org/downloads/mysql-2014-04-02.sql.gz
Chapter 4

Constructing the Graphs

To investigate socio-technical congruence, we need to be able to form graphs that represent both technical and social dependences that exist between the selected 250 projects. We analyze the GitHub and Maven data to form two graphs: a technical dependence graph (Section 4.1) and a social dependence graph (Section 4.2).

4.1 Technical Dependence Graph

To represent technical dependencies, we form a technical dependence graph $TDG = (V_P, E_{TDG})$. Let $P$ be the set of projects, then $V_P$ is the set of vertices where each project corresponds to a vertex. The edge set, $E_{TDG}$, defines directed edges between vertices in the graph, one for each technical dependence.

Let $L$ be the set of libraries from all projects; each project can have one or more libraries, and let $D$ be the set of dates. For each edge $e \in E_{TDG}$, we define $dependence(e) : E_{TDG} \rightarrow (L \times D)$. In other words, each edge is annotated with the date the technical dependence was introduced and the technical dependence at the time. Figure 4.1 shows an example technical dependence graph.

We determine $E_{TDG}$ based on the dependencies specified in the POM files extracted from the git repositories of the selected projects. We cloned git repositories for all of the selected projects on 2 April 2014 and then extracted all POM files at the latest commit in each project. To identify a GitHub project uniquely we consider only groupId from the POM files, this is because a project can produce
multiple artefacts and for the purpose of this thesis, it does not matter which artefact of a project introduces a technical dependence. However, we do look over the repository history and keep track of the time a technical dependence was introduced and the technical dependence at that time, each edge is annotated with this information.

4.2 Social Project Graph

The social project graph (SPG) describes interactions between projects. This graph aggregates interactions between GitHub users\(^1\) via issues and other mechanisms to the project level to enable comparison to the TDG. We say that an interaction has

\(^1\)We use the term *user* as opposed to *developer* since a GitHub user may only open issues or comment on issues and never write code. However, it is likely that a majority of GitHub *users* do in fact write code and can be considered *developers*.
occurred between two users when they have performed an action on the same issue, or the same pull request. We consider three actions on an issue: create, comment, and close. We consider four actions on a pull request: create, comment, merge, and close. The issues and pull requests belong to projects. We say that two projects interact with each other if users who work on these projects interact with each other. Hence, to construct the SPG, we aggregate these user-user interactions to project-project interactions.

4.2.1 User Interaction Graph

We first construct an undirected multigraph which captures user interactions $UIG = (V_U, E_{UIG})$. Let $U$ be the set of users who have at least one action on an issue or a pull request, then $V_U$ is the set of vertices where each user corresponds to a vertex. The multiset, $E_{UIG}$, defines undirected edges between vertices in the graph. Each edge corresponds to interactions between two users on a single project. If two users interact on multiple projects, there are multiple edges between the corresponding vertices, one for each project.

Let $P$ be the set of projects. For each edge $e \in E_{UIG}$, we define $\text{project}(e) : E_{UIG} \rightarrow P$. In other words, we keep track of specific projects by annotating each edge with the project to which the corresponding user interactions belong.

Let $x$ and $y$ be two users and $p$ a project. For each edge $e_{xyp} \in E_{UIG}$, we define $\text{weight}(e_{xyp}) : E_{UIG} \rightarrow \mathbb{N}$.

The weight function is given by

$$\text{weight}(e_{xyp}) = \sum_{i_{xyp} \in i_{xyp}} (|i_{xyp}(x)| \times |i_{xyp}(y)|) + \sum_{r_{xyp} \in r_{xyp}} (|r_{xyp}(x)| \times |r_{xyp}(y)|)$$
where

\[ x \] and \( y \) are two users
\[ p \] is a project
\[ I_{xyp} \] is the set of issues on \( p \) on which both \( x \) and \( y \) performed an action
\[ R_{xyp} \] is the set of pull requests on \( p \) on which both \( x \) and \( y \) performed an action
\[ i_{xyp}(x) \] is the set of actions of user \( x \) on issue \( i_{xyp} \)
\[ r_{xyp}(x) \] is the set of actions of user \( x \) on pull request \( r_{xyp} \)

In other words, each edge is weighted by the number of times the two corresponding users interacted with each other on a single project. Figure 4.2 shows an example user interaction graph.

![User Interaction Graph](image)

**Figure 4.2:** An example user interaction graph with only two users \( x, y \). These users interact on three projects \( p, q, n \). Each edge is annotated with the project to which the corresponding user interactions belong, and each edge is weighted by the number of times the two users interacted with each other on that project.

The UIG formed from the dataset has 18,471 vertices and 90,837 edges. On average, a user has interacted with five other users across all projects. Figure 4.3 shows the connectedness of users in UIG. Majority of the users have interacted with less than 10 other users. However, there are a few who have interacted with a larger (up to 50) number of users.

The UIG results in an over approximation of actual interactions as the com-
munication might have been one-way. Different users comment on an artefact at different times, possibly weeks or months apart. Hence, we cannot be sure of an interaction between all users who commented on an artefact, since we don’t know if a comment was read by another user. In some ways, the UIG is also an underestimation because not all communication related to projects may be captured in a repository. We discuss these issues further in Chapter 6.

4.2.2 Forming the Social Project Graph

To represent social interactions between projects, we form a social project graph $SPG = (V_P, E_{SPG})$. Let $P$ be the set of projects, then $V_P$ is the set of vertices where each project corresponds to a vertex. The edge set, $E_{SPG}$, defines undirected edges between vertices in the graph. Each edge corresponds to social interactions between two projects.

Let $U_{mn}$ be the set of users who performed an action on some issues or pull requests of projects $m$ and $n$. Also, let $u_{mn} \in U_{mn}$ and let $IA_{mn}(u_{mn})$ be set of issue actions and $RA_{mn}(u_{mn})$ be set of pull request actions performed by user $u_{mn}$ on

![Figure 4.3: Degree of nodes in UIG](image)

Figure 4.3: Degree of nodes in UIG
projects $m$ and $n$. Then for each edge $e \in E_{SPG}$, we define

$$\text{actions}(e_{mn}) = \bigcup_{u_{mn} \in U_{mn}} \left( I_{mn}(u_{mn}) \bigcup R_{mn}(u_{mn}) \right)$$

In other words, each edge is annotated with all the actions of the users on the two projects who were active on both the projects. Figure 4.4 shows an example social project graph.

![Figure 4.4: An example social project graph with four projects p, q, m, n. Each edge is annotated with all the actions of the users on the two projects who were active on both the projects.](image)

To form the SPG, we need to determine projects that interact with each other based on the UIG. We only want to represent interactions between projects if there are a number of interactions between users working on those projects. To determine where there are strong interactions between users, we apply community detection to the UIG. Girvan and Newman defined community detection as the division of a graph into communities or sub-graphs in which the connections within communities are much denser than the connections between them [10]. By definition community detection filters out weak connections between users, and hence
Figure 4.5: Community detection in user interaction graph

allows us to focus on strong interactions. Since we say that two projects interact with each other if users who work on these projects interact with each other, we construct the SPG by applying community detection to the UIG and forming the SPG from the result by forming edges between all projects in a community.

Figure 4.5 provides an illustration of community structure in a graph. This figure shows three communities with dense connections within each community, but only a single connection between communities. In Figure 4.5 each edge is also labelled with the project name corresponding to the interactions. For this graph we create an SPG with vertices \{p, q, m, n\} and edges \{pq, pn, qn, mn\}. This is the graph shown in Figure 4.4.

Forming edges between all projects in a community results in an over approximation of project to project social interactions. The rationale of the SPG is to capture potential indirect interactions between people interacting on different projects much like social network analysis. We discuss this more in Chapter 6.

We use the fast community detection algorithm by Blondel et al. [3]. This algorithm is a heuristic method that is based on modularity optimization. It outperforms all other known community detection method in terms of computation time. At the same time, the quality of the communities detected is also good as
measured by *modularity*. The modularity of detected communities is a measure to quantify the goodness of detected communities and it is a scalar value between -1 and 1 that measures the density of links inside communities as compared to links between communities [16]. We use the implementation available in the Python *igraph*\(^2\) library.

When applied to the dataset, the algorithm detects 104 communities in the UIG.

\(^2\)http://igraph.sourceforge.net/
Figure 4.7: Example communities detected in UIG. Nodes are users and the edges are the interaction between users. Different edge colors within a community represent the different projects over which the interactions occurred.

Figure 4.6a shows a histogram of number of projects involved in each community detected. From this histogram, we can see that a large number of communities involve interactions on a single project, and most communities involve eight projects or less. Figure 4.6b shows a histogram of the number of users involved in each community; most communities are quite small. Figure 4.7 shows a few communities detected in UIG. Each vertex corresponds to a user and each edge corresponds to interactions between two users on a single project. Different edge colors in a single community correspond to different projects the users in the community
interact on. Some communities involve interactions on only one project (e.g., Figure 4.7a). Some communities connect two projects, but there may be only one connecting user (e.g., Figure 4.7b). Other communities may have several connecting users (e.g., Figure 4.7c). There are also some communities which connect several projects and there are several connecting users (e.g. Figure 4.7d).
Chapter 5

Results

We consider each research question in turn.

5.1 Socio-Technical Congruence in an Ecosystem

To answer our first research question, "How often does congruence between technical dependencies and social interactions occur in an ecosystem of software developers?", we compare the TDG and the SPG formed from the data of the 250 GitHub projects described in Chapter 4. The number of project pairs that have some social interaction is 1809, in comparison, the number of project pairs related by a technical dependence is 664. The overlap between the TDG and the SPG is 157 edges, meaning that in 23.6% of cases in which there is a technical dependence between projects there is also evidence of social interaction between project members. However, in only 8.67% of cases in which there is a social interaction between project members, there is a technical dependence between projects.

To provide a sense of the graphs, we compare the connectedness of nodes in the TDG and the SPG. Figure 5.1a shows the degree of TDG vs degree of SPG; since TDG is directed Figure 5.1b shows the in-degree of TDG vs degree of SPG. Each dot in these plots corresponds to a project. Figure 5.1 shows that most projects have less than 10 technical dependencies to other selected projects from GitHub, however a number of projects interact with up to 50 other projects.

Table 5.1 summarizes these numbers.
### Table 5.1: Overview of TDG and SPG

<table>
<thead>
<tr>
<th>#nodes in both SPG and TDG</th>
<th>250</th>
</tr>
</thead>
<tbody>
<tr>
<td>#edges in TDG</td>
<td>664</td>
</tr>
<tr>
<td>#edges in SPG</td>
<td>1809</td>
</tr>
<tr>
<td>avg in-degree of TDG</td>
<td>2.656</td>
</tr>
<tr>
<td>avg degree of SPG</td>
<td>14.472</td>
</tr>
<tr>
<td>#edges overlapping in TDG and SPG</td>
<td>157</td>
</tr>
<tr>
<td>#edges TDG-SPG</td>
<td>507</td>
</tr>
<tr>
<td>#edges SPG-TDG</td>
<td>1652</td>
</tr>
</tbody>
</table>

### 5.2 Origins of Socio-Technical Congruence

To answer our second research question, "When socio-technical congruence does exist, how does it come about?", we consider the common edges in the TDG and the SPG. For each overlapping edge, we determine all users who contributed to both the projects and extract the activity of each such user on these two projects (i.e., \( \text{actions}(e_{mn}) \) as defined in Section 4.2.2). The activity information provides us the project, user, type of interaction and time at which the action occurred. Since we over approximate the number of edges in the SPG, for some of the overlapping edges there are no common users. The occurrences of over approximation are small, occurring 18 times out of 157 (11.46%). For each overlapping edge, we also extract the time when a technical dependence was introduced between the two projects (i.e., \( \text{dependence}(e) \) as defined in Section 4.1).

For each project pair, we plot the activity of each user in common to both projects of the pair as a timeline to understand if there are patterns that occur. Figures 5.3, 5.4, and 5.5 show three such plots. The x-axis shows the time period under consideration, and the y-axis shows different users who are active on both projects, the labels on y-axis are the user ids of these users in the GHTorrent dataset. Each square corresponds to a user action on an issue and a circle corresponds to a user action on a pull request. For each user, there are two timelines, one for each project that are distinguished by colour. The data points in blue correspond to the first project in the figure title and the data points in green correspond to the second project. In all cases, the first project has a technical dependence on the second project. The date the technical dependence was introduced is shown by
We also plot the percentage of social activity that happens after the technical dependence of interest is introduced. In Figure 5.2, each dot corresponds to a project pair. We categorize all project pairs in three categories:

A) All social activity for users who contribute to both projects comes after the technical dependence is introduced. There are 45 such project pairs.
B) More than 75% of social activity comes after technical dependence is introduced. There are 49 such project pairs.

C) The remaining 45 project pairs, in which less than 75% of social activity comes after the technical dependence.

We chose a threshold of 75% to split the plots into category B and C into similar size groupings.

5.2.1 Qualitative Examination

To better understand the categories, we visually examined the timeline plots for each category and selected a project pair that is representative of the category. For each such project pair, we search for textual match for the dependency name in all the issues and pull requests for the project creating the dependency as a means of determining issues and pull requests related to the dependence. After this filtering process, we manually examine each issue and pull request and all the comments to understand more about the technical dependency. We present three such selected examples below.
Category A: All social activity after technical dependence introduction. The example for Category A is the project pair: wildfly/wildfly\(^1\) and resteasy/Resteasy\(^2\).

The WildFly Application server, formerly known as JBoss AS\(^3\) (JBoss Application Server), is a flexible, lightweight, managed application runtime. It implements the Java Platform, Enterprise Edition (Java EE) specification. RESTEasy is a JBoss.org project aimed at providing productivity frameworks for developing client and server RESTful applications and services in Java. It is a fully certified and portable implementation of the JAX-RS specification. JAX-RS is a new JCP specification that provides a Java API for RESTful Web Services over the HTTP protocol.

WildFly introduced a dependency on RESTEasy on Feb 25 2011. The first mention of the technical dependency occurs six months later in August 2011 about upgrading the dependency from version 2.2.1 GA to version 2.2.2 GA\(^4\). There are

---

1. https://github.com/wildfly/wildfly
2. https://github.com/resteasy/Resteasy
six further mentions of upgrading the dependency version in Jan 2013\(^5\), Feb 2013\(^6\), June 2013\(^7\), Sep 2013\(^8,9\), Oct 2013\(^10\) and Dec 2013\(^11\). All these upgrades involved creating a build with the new version and then running tests. If the build succeeded the dependency version was upgraded. On occasion, there is mention of the reason for upgrading to a new version. For instance, the upgrade in Feb 2013 happens because of need for *JAX-RS 2.0 support*, and the upgrade in June 2013 happens to be able to support *multiple Application classes*.

Over time more dependencies to RESTEasy are added. In Jan 2012 a dependency to resteasy-yaml-provider is added\(^12\). In May 2013 a dependency to resteasy-crypto module is added\(^13\). In July 2013 a dependency to resteasy-client module is added\(^14\). In Jan 2014 a dependency to resteasy-spring is added\(^15\).

There are several other issues which talk about specific bugs or adding more tests.

*Category B: 75% social activity comes after technical dependence introduction.* The example for Category B is the project pair: *infinispan/infinispan*\(^16\) and *weld/core*\(^17\).

Infinispan is an open source data grid platform and highly scalable NoSQL cloud data store. Weld is the reference implementation of CDI: Contexts and Dependency Injection for the Java EE Platform which is the Java standard for dependency injection and contextual lifecycle management. Weld is integrated into many Java EE application servers such as WildFly, JBoss Enterprise Application Platform, GlassFish, Oracle WebLogic and others.

\(^5\)https://github.com/wildfly/wildfly/pull/3838  
\(^6\)https://github.com/wildfly/wildfly/pull/4117  
\(^7\)https://github.com/wildfly/wildfly/pull/4673  
\(^8\)https://github.com/wildfly/wildfly/pull/4991  
\(^9\)https://github.com/wildfly/wildfly/pull/5009  
\(^10\)https://github.com/wildfly/wildfly/pull/5391  
\(^11\)https://github.com/wildfly/wildfly/pull/5596  
\(^12\)https://github.com/wildfly/wildfly/pull/1195  
\(^13\)https://github.com/wildfly/wildfly/pull/4565  
\(^14\)https://github.com/wildfly/wildfly/pull/4758  
\(^15\)https://github.com/wildfly/wildfly/pull/5695  
\(^16\)https://github.com/infinispan/infinispan  
\(^17\)https://github.com/weld/core
Infinispan introduced a dependency on Weld on Jul 18 2011. Five users who eventually contribute to both projects are active on at least one project before this date. Two of them have contributions on both projects, and one of them *pmuir*\(^{18}\) has multiple contributions on both projects prior to the introduction of technical dependence. The technical dependence is introduced by a pull request\(^{19}\). *pmuir* is the author on some commits in this pull request, and also the one who merges the pull request. There is not however, a discussion on merits or demerits of adding this technical dependence. Apart from this pull request *pmuir*’s contributions to both projects are unrelated to the technical dependence.

After the dependency is introduced several pull requests are opened to make sure that the version of Weld being used by Infinispan is kept up to date in Sept

---

\(^{18}\)https://github.com/pmuir

\(^{19}\)https://github.com/infinispan/infinispan/pull/420
2011\textsuperscript{20}, Nov 2011\textsuperscript{21}, Dec 2011\textsuperscript{22}, Nov 2012\textsuperscript{23} and June 2013\textsuperscript{24}. This last one in June 2013 is opened by \textit{pmuir}. In this pull request updating the dependence is a small part, the main focus of this is to do a code cleanup.

\textbf{Figure 5.5:} Category C-sarxos/webcam-capture(blue) and netty/netty(green)

\textit{Category C: Less than 75\% of social activity after technical dependence introduction.} The example of Category C is the project pair: \texttt{sarxos/webcam-capture}\textsuperscript{25} and \texttt{netty/netty}\textsuperscript{26}.

The project sarxos/webcam-capture is a Webcam Capture API for Java. This library allows one to use the build-in or external webcam directly from Java. It’s designed to abstract commonly used camera features and support multiple captur-
ing frameworks. Netty is an asynchronous event-driven network application framework for rapid development of maintainable high performance protocol servers and clients.

Webcam-capture introduced a dependency on Netty in Mar 2013. One user hepin1989 creates a pull-request on webcam-capture project for a Webcam Capture live streaming example. Another user sarxos accepts this pull request - "Thank you :) I really appreciate your help. Tomorrow I will Mavenize it.". This pull request introduces the technical dependency.

Prior to creating this pull request, this user hepin1989 also opens multiple issues on both projects starting Jan 2013. These issues were bugs the user encountered while trying to use both the projects. On one such issue the following exchange takes place which eventually results in the technical dependency creation.

ZAHIDHAF: can you please send me the code webcam encode and decode using xuggler.

HEPIN1989: i will send you an gist [hepin1989 sends a gist]

SARXOS: Can I refine your code and include it as a new usage example?

HEPIN1989: I will write example for your project, how about this?

SARXOS: It would be perfect if you prepare such example, thank you :) 
HEPIN1989: Ok, I will going to write something about it and could i use netty, or just the original java NIO?

SARXOS: Sure :) Please use whatever you want until it is available from Maven Central.

After the pull request is accepted, there is some follow up discussion on using the Webcam Capture live streaming example with an Android phone as a client. However, since this technical dependency was introduced for one specific piece of code, after that feature is implemented and works well there isn’t much social interaction.

---

27 https://github.com/hepin1989
28 https://github.com/sarxos/webcam-capture/pull/68
29 https://github.com/sarxos
30 https://github.com/sarxos/webcam-capture/issues/11
31 https://github.com/sarxos/webcam-capture/pull/68#issuecomment-19867443
Chapter 6

Discussion

We made a number of choices in the TDG and SPG formation in our investigation of socio-technical congruence. We discuss these choices and also consider our overall approach for investigating socio-technical congruence.

6.1 Formulations of TDG

We form the TDG by identifying technical dependencies at the level of Java libraries. However, we do not know if a library is actually used by a project or how often it is used. Hence, an argument could be made to look for technical dependencies at a more fine grained level of import statements, since such analysis would more accurately capture library usage. However, such a fine grained analysis would be quite expensive computationally, since it requires scanning all the source files. For this exploratory study, we chose the computationally cheaper option, and we leave a more fine grained analysis of technical dependencies as future work.

While forming the TDG, we assume that the POM files if present in a git repository specify the technical dependencies completely and correctly. However, we never compile the projects to ascertain if this is in fact true. It is possible that we miss library jar files may have been copied in, or a project does not actually use POM files for dependency management.
6.2 Formulations of SPG

In the UIG formation, we give equal weights to all types of interactions, i.e. we do not differentiate between interactions on pull requests and interactions on issues.

The *weight* function from Section 4.2.1 could be changed to

\[
weight(e_{xyp}) = weight_i \times \sum_{i_{xyp} \in I_{xyp}} (|i_{xyp}(x)| \times |i_{xyp}(y)|) \\
+ weight_{pr} \times \sum_{r_{xyp} \in R_{xyp}} (|r_{xyp}(x)| \times |r_{xyp}(y)|)
\]

where

- *weight* \(_i\) is the weight of an interaction on an issue
- *weight* \(_{pr}\) is the weight of an interaction on a pull request

Experimenting with different weights is left as future work. In this thesis, we give equal weights to different types of interactions, i.e. *weight* \(_i\) = *weight* \(_{pr}\) = 1.

The user interaction graph (UIG) potentially underestimates user to user interactions because not all communication related to projects may be captured in the GitHub repository. In this work, we do not consider communication between users that may happen outside of GitHub issues, such as email exchanges and chat conversations between users or mailing lists that may be used by projects. However, this is acceptable because social network analysis in open source projects is still valid in case of missing or inadequate data. Nia et al.[17] studied the effect of missing links and temporal data aggregation on measures of centrality of nodes, including clustering coefficient, in the network. They demonstrate on three different OSS projects that while these issues do change network topology, the measures are stable with respect to such changes.

From another perspective, the UIG may also be an overapproximation of actual interactions in GitHub. We assume that two users interacted if they commented on the same issue or a pull request. However, one user may not have read a comment by another user. However, this is reasonable because there are a small number of
comments on average on issues and pull requests. The average number of comments on an issue is 2.14 and on a pull request is 0.74 (Table 3.1). The small number of comments on each issue and pull request imply that on an average we discover a small number of user-user interactions per issue or pull request, which reduces the amount of over approximation.

To form the SPG, we first detect user communities in the UIG. There are other possible approaches to creating a SPG. For instance, we could simply form connections between all those projects that have at least one common user as a contributor. However, this approach could result in connections between projects where a single user interacted with both projects just once. This kind of single isolated connection would probably represent noise rather than actual connection between two projects. To overcome this problem, we could have selected minimum thresholds for number of common users and number of interactions a user needs to have with a project. However, these minimum thresholds would be arbitrary. Hence, we chose to go with community detection, since that provides us a mechanism to filter out the weak interactions.

We did not validate the detected communities with actual users because it is not necessary for the communities found by the community detection algorithm to be recognizable to a user. We use community detection algorithm solely as a means to filter out weak connections between users.

When we form the SPG, we form edges between all projects in a community. This approach results in an over approximation of project to project social interactions. Later, for all edges common to both the TDG and the SPG, we examine the activity of all users who contributed to both projects. However, because of the over approximation, for some of the overlapping edges, there are no common users. The number of edges for which this is true is small, 18 out of 157 or 11.46%. As a result, to understand how far apart these projects might be in UIG, we want to get a sense of distance between users in a community. Figure 6.1 shows the distances between users in a community in UIG, both the average distances and the largest distances. The figure on the left shows that for most users are on an average two users away from any other user. The figure on the right shows that in most communities the largest distance between two users is five.
6.3 Understanding Socio-Technical Congruence

When comparing the TDG and the SPG, we look for the existence of a connection between two projects in both graphs. However, we ignore the strength of this connection. The strength of a connection in the TDG could be related to the number of libraries from one project on which the other project depends. In terms of the SPG, the strength of a connection could be related to the number of users who
contribute to both projects. Investigating connection strength is left to future work.

To understand the origins of socio-technical congruence, we investigate a few project pairs manually. More sophisticated methods of automatic analysis could be used to classify issue and pull request comments based on their relationship to the technical dependence. For instance, automatic analysis could be used to investigate discussions about a dependence, bugs related to dependences and dependence upgrades. Applying such an analysis would give a more thorough understanding of the categories of how social and technical congruences occur.
Chapter 7

Conclusion

We explored socio-technical congruence in 250 projects on GitHub that use Maven for build dependences. We created a social interaction graph based on developers’ interactions on issue and pull requests, and we used community detection techniques to identify strong user-user interactions. We also created a technical dependence graph based on build dependences specified in POM files used by Maven. We compared these two graphs to get an overview of the congruence, or lack thereof, between social interactions and technical dependences. We found that for 23.6% of the cases in which there is a technical dependence between two projects, there is also evidence of some social interactions. In only 8.67% of the cases when there is a social interaction between projects is there also a technical dependence.

We categorized all cases in which there are both social and technical dependences between projects. We found that for 45 (32%) of project pairs, no social interaction had taken place before the introduction of technical dependence and interactions after the introduction of the dependence are often about upgrading the library being depended upon. For 49 (35%) of project pairs, 75% of the interaction takes place after the introduction of the technical dependence. For the remaining 45 (32%) of project pairs, less than 75% of the interaction takes place after the introduction of the technical dependence. In the latter two cases, although there is interaction before the technical dependence is introduced, it is not always about the dependence.
We also discussed some of the ways that this exploratory study could be fine tuned in the future.
Bibliography


Appendix A

Projects Explored

The 250 GitHub projects explored in this dissertation:

1. wildfly/wildfly
2. elasticsearch/elasticsearch
3. netty/netty
4. infinispan/infinispan
5. brooklyncentral/brooklyn
6. jcabi/jcabi-github
7. getlantern/lantern
8. Bukkit/CraftBukkit
9. MasDennis/Rajawali
10. neo4j/neo4j
11. openmicroscopy/bioformats
12. mcMMO-Dev/mcMMO
13. hazelcast/hazelcast
14. junit-team/junit
15. turesheim/eclipse-utilities
16. jbosstm/narayana
17. Multiverse/Multiverse-Core
18. mff-uk/ODCS
19. hibernate/hibernate-orm
20. cucumber/cucumber-jvm
<table>
<thead>
<tr>
<th></th>
<th>Repository Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Catrobat/Catroid</td>
</tr>
<tr>
<td>22</td>
<td>Bukkit/Bukkit</td>
</tr>
<tr>
<td>23</td>
<td>nathanmarz/storm</td>
</tr>
<tr>
<td>24</td>
<td>FamilySearch/gedcomx</td>
</tr>
<tr>
<td>25</td>
<td>square/okhttp</td>
</tr>
<tr>
<td>26</td>
<td>AndlyticsProject/andlytics</td>
</tr>
<tr>
<td>27</td>
<td>irstv/orbisgis</td>
</tr>
<tr>
<td>28</td>
<td>gradle/gradle</td>
</tr>
<tr>
<td>29</td>
<td>forcedotcom/phoenix</td>
</tr>
<tr>
<td>30</td>
<td>ArcBees/GWTP</td>
</tr>
<tr>
<td>31</td>
<td>hornetq/hornetq</td>
</tr>
<tr>
<td>32</td>
<td>ujhelyiz/EMF-IncQuery</td>
</tr>
<tr>
<td>33</td>
<td>BroadleafCommerce/BroadleafCommerce</td>
</tr>
<tr>
<td>34</td>
<td>caelum/vraptor4</td>
</tr>
<tr>
<td>35</td>
<td>stratosphere/stratosphere</td>
</tr>
<tr>
<td>36</td>
<td>molgenis/molgenis</td>
</tr>
<tr>
<td>37</td>
<td>dana-i2cat/opennaas</td>
</tr>
<tr>
<td>38</td>
<td>hector-client/hector</td>
</tr>
<tr>
<td>39</td>
<td>square/dagger</td>
</tr>
<tr>
<td>40</td>
<td>gxa/gxa</td>
</tr>
<tr>
<td>41</td>
<td>mozilla-services/android-sync</td>
</tr>
<tr>
<td>42</td>
<td>FoundationDB/sql-layer</td>
</tr>
<tr>
<td>43</td>
<td>square/picasso</td>
</tr>
<tr>
<td>44</td>
<td>Sage-Bionetworks/SynapseWebClient</td>
</tr>
<tr>
<td>45</td>
<td>pentaho/pentaho-platform</td>
</tr>
<tr>
<td>46</td>
<td>weld/core</td>
</tr>
<tr>
<td>47</td>
<td>jboss-reddeer/reddeer</td>
</tr>
<tr>
<td>48</td>
<td>zanata/zanata-server</td>
</tr>
<tr>
<td>49</td>
<td>square/retrofit</td>
</tr>
<tr>
<td>50</td>
<td>owncloud/android</td>
</tr>
<tr>
<td>51</td>
<td>DataTorrent/Malhar</td>
</tr>
<tr>
<td>52</td>
<td>thinkaurelius/titan</td>
</tr>
<tr>
<td>53</td>
<td>intermine/intermine</td>
</tr>
</tbody>
</table>
54. rackerlabs/repose  
55. SpigotMC/BungeeCord  
56. nostra13/Android-Universal-Image-Loader  
57. ceylon/ceylon-ide-eclipse  
58. metamx/druid  
59. cwi-swat/rascal  
60. imeji-community/imeji  
61. bndtools/bnd  
62. Caleydo/caleydo  
63. essentials/Essentials  
64. fabric8io/fabric8  
65. rackerlabs/blueflood  
66. SpoutDev/Spout  
67. jfeinstein10/SlidingMenu  
68. JakeWharton/ActionBarSherlock  
69. geoserver/geoserver  
70. atlasapi/atlas  
71. jim618/multibit  
72. hibernate/hibernate-ogm  
73. jboss-fuse/fuse  
74. kevinweil/elephant-bird  
75. jacoco/jacoco  
76. l0rdn1kk0n/wicket-bootstrap  
77. endosnipe/ENdoSnipe  
78. tinkerpop/rexster  
79. BaseXdb/basex  
80. facebook/presto  
81. Prototik/HoloEverywhere  
82. lobid/lobmill  
83. richardwilly98/elasticsearch-river-mongodb  
84. UniversalMediaServer/UniversalMediaServer  
85. jboss-switchyard/core  
86. overturerepo/overture
87. sk89q/worldedit
88. OpenGrok/OpenGrok
89. maxcom/lorsource
90. griddynamics/jagger
91. restlet/restlet-framework-java
92. xetorthio/jedis
93. nutzam/nutz
94. droolsjbpm/jbpm
95. caelum/vraptor
96. sarxos/webcam-capture
97. danieloeh/AntennaPod
98. MythTV-Clients/MythTV-Android-Frontend
99. amplab/tachyon
100. openlegacy/openlegacy
101. kijiproject/kiji-schema
102. jbosstools/jbosstools-integration-tests
103. crawljax/crawljax
104. selendroid/selendroid
105. Graylog2/graylog2-server
106. caprica/vlcj
107. filipg/amu_automata_2011
108. alibaba/RocketMQ
109. itm/testbed-runtime
110. e-ucm/cad
111. laforge49/JActor2
112. Multiverse/Multiverse-Portals
113. SurvivalGamesDevTeam/TheSurvivalGames
114. nla/banjo
115. jberkel/sms-backup-plus
116. rgladwell/m2e-android
117. curtisullerich/attendance
118. EmilHernvall/tregmine
119. librefiodev/android
<table>
<thead>
<tr>
<th>No.</th>
<th>Repository Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>mapstruct/mapstruct</td>
</tr>
<tr>
<td>121</td>
<td>Monstercraft/MonsterIRC</td>
</tr>
<tr>
<td>122</td>
<td>gwtquery/gwtquery</td>
</tr>
<tr>
<td>123</td>
<td>basho/riak-java-client</td>
</tr>
<tr>
<td>124</td>
<td>jboss-switchyard/quickstarts</td>
</tr>
<tr>
<td>125</td>
<td>jboss-switchyard/tools</td>
</tr>
<tr>
<td>126</td>
<td>Adobe-Consulting-Services/acs-aem-commons</td>
</tr>
<tr>
<td>127</td>
<td>jbosstools/jbosstools-server</td>
</tr>
<tr>
<td>128</td>
<td>springside/springside4</td>
</tr>
<tr>
<td>129</td>
<td>apigee/usergrid-stack</td>
</tr>
<tr>
<td>130</td>
<td>Silverpeas/Silverpeas-Components</td>
</tr>
<tr>
<td>131</td>
<td>graphhopper/graphhopper</td>
</tr>
<tr>
<td>132</td>
<td>mitreid-connect/OpenID-Connect-Java-Spring-Server</td>
</tr>
<tr>
<td>133</td>
<td>DSH105/EchoPet</td>
</tr>
<tr>
<td>134</td>
<td>alibaba/druid</td>
</tr>
<tr>
<td>135</td>
<td>jayway/maven-android-plugin</td>
</tr>
<tr>
<td>136</td>
<td>undertow-io/undertow</td>
</tr>
<tr>
<td>137</td>
<td>dCache/dcache</td>
</tr>
<tr>
<td>138</td>
<td>resteasy/Resteasy</td>
</tr>
<tr>
<td>139</td>
<td>samarthgupta437/falcon-regression</td>
</tr>
<tr>
<td>140</td>
<td>bigbluebutton/bigbluebutton</td>
</tr>
<tr>
<td>141</td>
<td>CruGlobal/conf-registration-api</td>
</tr>
<tr>
<td>142</td>
<td>demoiselle/behave</td>
</tr>
<tr>
<td>143</td>
<td>irstv/H2GIS</td>
</tr>
<tr>
<td>144</td>
<td>Findwise/Hydra</td>
</tr>
<tr>
<td>145</td>
<td>yegor256/s3auth</td>
</tr>
<tr>
<td>146</td>
<td>NightWhistler/Page Turner</td>
</tr>
<tr>
<td>147</td>
<td>huskysoft/403Interviewer</td>
</tr>
<tr>
<td>148</td>
<td>nasa/mct</td>
</tr>
<tr>
<td>149</td>
<td>branflake2267/GWT-Maps-V3-Api</td>
</tr>
<tr>
<td>150</td>
<td>ios-driver/ios-driver</td>
</tr>
<tr>
<td>151</td>
<td>jclouds/jclouds-chef</td>
</tr>
<tr>
<td>152</td>
<td>sk89q/commandhelper</td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>153.</td>
<td>Cloudname/cloudname</td>
</tr>
<tr>
<td>154.</td>
<td>mbax/VanishNoPacket</td>
</tr>
<tr>
<td>155.</td>
<td>xXKeyleXx/MyPet</td>
</tr>
<tr>
<td>156.</td>
<td>jbosstools/jbosstools-base</td>
</tr>
<tr>
<td>157.</td>
<td>jbosstm/quickstart</td>
</tr>
<tr>
<td>158.</td>
<td>github/android</td>
</tr>
<tr>
<td>159.</td>
<td>MinecraftPortCentral/MCPC-Plus-Legacy</td>
</tr>
<tr>
<td>160.</td>
<td>pulse00/Symfony-2-Eclipse-Plugin</td>
</tr>
<tr>
<td>161.</td>
<td>FenixEdu/fenix</td>
</tr>
<tr>
<td>162.</td>
<td>Jasig/uPortal</td>
</tr>
<tr>
<td>163.</td>
<td>The-Dream-Team/Tardis</td>
</tr>
<tr>
<td>164.</td>
<td>liquibase/liquibase</td>
</tr>
<tr>
<td>165.</td>
<td>jclouds/jclouds-karaf</td>
</tr>
<tr>
<td>166.</td>
<td>dbpedia-spotlight/dbpedia-spotlight</td>
</tr>
<tr>
<td>167.</td>
<td>jankotek/MapDB</td>
</tr>
<tr>
<td>168.</td>
<td>jbosstools/jbosstools-openshift</td>
</tr>
<tr>
<td>169.</td>
<td>jboss-switchyard/release</td>
</tr>
<tr>
<td>170.</td>
<td>yahoo/oozie</td>
</tr>
<tr>
<td>171.</td>
<td>vkostyukov/la4j</td>
</tr>
<tr>
<td>172.</td>
<td>Hidendra/LWC</td>
</tr>
<tr>
<td>173.</td>
<td>OpenNTF/JavascriptAggregator</td>
</tr>
<tr>
<td>174.</td>
<td>jbosstools/jbosstools-central</td>
</tr>
<tr>
<td>175.</td>
<td>NineWorlds/serenity-android</td>
</tr>
<tr>
<td>176.</td>
<td>webbit/webbit</td>
</tr>
<tr>
<td>177.</td>
<td>maplesyrup/maple-android</td>
</tr>
<tr>
<td>178.</td>
<td>rydnr/queryj</td>
</tr>
<tr>
<td>179.</td>
<td>robovm/robovm</td>
</tr>
<tr>
<td>180.</td>
<td>ceylon/ceylon-module-resolver</td>
</tr>
<tr>
<td>181.</td>
<td>p6spy/p6spy</td>
</tr>
<tr>
<td>182.</td>
<td>karma-exchange-org/karma-exchange</td>
</tr>
<tr>
<td>183.</td>
<td>fixteam/fixflow</td>
</tr>
<tr>
<td>184.</td>
<td>square/wire</td>
</tr>
<tr>
<td>185.</td>
<td>jbosstools/jbosstools-jst</td>
</tr>
</tbody>
</table>
186. Jasig/java-cas-client
187. mybatis/mybatis-3
188. redline-smalltalk/redline-smalltalk
189. taoneill/war
190. OSBI/saiku
191. greenlaw110/Rythm
192. eclipse/vert.x
193. Activiti/Activiti
194. symphonytool/symphony
195. Nodeclipse/nodeclipse-1
196. axemblr/axemblr-provisionr
197. MxUpdate/Update
198. eclipsesource/tabris
199. mongodb/morphia
200. tcurdt/jdeb
201. aws/aws-sdk-java
202. javajigi/slipp
203. docdoku/docdoku-plm
204. jenkinsci/git-client-plugin
205. capedwarf/capedwarf-blue
206. TooTallNate/Java-WebSocket
207. jantje/arduino-eclipse-plugin
208. joel-costigliola/assertj-core
209. MarkehMe/FactionsPlus
210. tntim96/JSCover
211. pardom/ActiveAndroid
212. chrisbanes/PhotoView
213. Governance/s-ramp
214. alkarinv/BattleArena
215. Microsoft-CISL/REEF
216. ralscha/extdirectspring
217. nhaarman/ListViewAnimations
218. SeqWare/seqware
219. greenlaw110/play-morphia
220. facebook/swift
221. cbeust/testng
222. alexruiz/fest-assert-2.x
223. robotoworks/mechanoid
224. dropwizard/dropwizard
225. np98765/BattleKits
226. todoroo/astrid
227. wuetherich/bundlemaker
228. jknack/handlebars.java
229. jline/jline2
230. ps3mediaserver/ps3mediaserver
231. moagrius/TileView
232. andrewphorn/ClassiCube-Client
233. janinko/ghprb
234. openplanets/scout
235. compbio-UofT/medsavant
236. sonyxperiadev/BacklogTool
237. Docear/Desktop
238. objectos/objectos-dojo
239. jbosstools/jbosstools-vpe
240. marytts/marytts
241. ceylon/ceylon-runtime
242. BatooOrg/BatooJPA
243. korpling/ANNIS
244. backmeup/backmeup-prototype
245. carrotsearch/randomizedtesting
246. bguerout/jongo
247. kijiproject/kiji-mapreduce
248. sqlparser/sql2jooq
249. aerogear/aerogear-android
250. CompendiumNG/CompendiumNG
Appendix B

Project Pairs With Both Technical Dependence and Social Interactions

The 139 project pairs which have common users. We categorize these projects in 3 categories in Section 5.2.

1. Governance/s-ramp ↔ jboss-switchyard/release
2. jclouds/jclouds-karaf ↔ jclouds/jclouds-chef
3. greenlaw110/Rythm ↔ alibaba/druid
4. ceylon/ceylon-runtime ↔ ceylon/ceylon-module-resolver
5. neo4j/neo4j ↔ rgladwell/m2e-android
6. np98765/BattleKits ↔ Bukkit/CraftBukkit
7. jbosstms/quickstart ↔ weld/core
8. jbosstms/quickstart ↔ jbosstms/narayana
9. jbosstms/quickstart ↔ wildfly/wildfly
10. pardom/ActiveAndroid ↔ rgladwell/m2e-android
11. undertow-io/undertow ↔ netty/netty
12. JakeWharton/ActionBarSherlock ↔ rgladwell/m2e-android
13. JakeWharton/ActionBarSherlock ↔ todoroo/astrid
14. JakeWharton/ActionBarSherlock ↔ square/wire
15. MarkehMe/FactionsPlus ↔ Bukkit/CraftBukkit
16. MarkehMe/FactionsPlus ↔ essentials/Essentials
17. weld/core ↔ cbeust/testng
18. weld/core ↔ wildfly/wildfly
19. weld/core ↔ fabric8io/fabric8
20. jboss-switchyard/release ↔ jboss-switchyard/quickstarts
21. bguerout/jongo ↔ joel-costigliola/assertj-core
22. BroadleafCommer/ce/BroadleafCommerce ↔ cbeust/testng
23. greenlaw110/play-morphia ↔ mongodb/morphia
24. webbit/webbit ↔ netty/netty
25. jboss-switchyard/tools ↔ jboss-switchyard/release
26. kijiproject/kiji-mapreduce ↔ kijiproject/kiji-schema
27. Bukkit/CraftBukkit ↔ jline/jline2
28. infospan/infinispan ↔ weld/core
29. infospan/infinispan ↔ resteasy/Resteasy
30. infospan/infinispan ↔ wildfly/wildfly
31. infospan/infinispan ↔ netty/netty
32. infospan/infinispan ↔ fabric8io/fabric8
33. jbosstm/narayana ↔ jbosstm/quickstart
34. jbosstm/narayana ↔ undertow-io/undertow
35. jbosstm/narayana ↔ weld/core
36. jbosstm/narayana ↔ wildfly/wildfly
37. p6spy/p6spy ↔ liquibase/liquibase
38. alkarinv/BattleArena ↔ Bukkit/CraftBukkit
39. alkarinv/BattleArena ↔ mbax/VanishNoPacket
40. alkarinv/BattleArena ↔ mcMMO-Dev/mcMMO
41. robovm/robovm ↔ rgladwell/m2e-android
42. square/retrofit ↔ rgladwell/m2e-android
43. square/retrofit ↔ square/wire
44. square/retrofit ↔ square/okhttp
45. mbax/VanishNoPacket ↔ Bukkit/CraftBukkit
46. SpoutDev/Spout ↔ jline/jline2
47. SpoutDev/Spout ↔ netty/netty
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>48.</td>
<td>jboss-switchyard/quickstarts ↔ weld/core</td>
</tr>
<tr>
<td>49.</td>
<td>jboss-switchyard/quickstarts ↔ jboss-switchyard/release</td>
</tr>
<tr>
<td>50.</td>
<td>jboss-switchyard/quickstarts ↔ jbosstm/narayana</td>
</tr>
<tr>
<td>51.</td>
<td>jboss-switchyard/quickstarts ↔ wildfly/wildfly</td>
</tr>
<tr>
<td>52.</td>
<td>chrisbanes/PhotoView ↔ nostra13/Android-Universal-Image-Loader</td>
</tr>
<tr>
<td>53.</td>
<td>mcMMO-Dev/mcMMO ↔ Bukkit/CraftBukkit</td>
</tr>
<tr>
<td>54.</td>
<td>alibaba/druid ↔ mybatis/mybatis-3</td>
</tr>
<tr>
<td>55.</td>
<td>alibaba/druid ↔ nutzam/nutz</td>
</tr>
<tr>
<td>56.</td>
<td>NineWorlds/serenity-android ↔ rgladwell/m2e-android</td>
</tr>
<tr>
<td>57.</td>
<td>resteasy/Resteasy ↔ weld/core</td>
</tr>
<tr>
<td>58.</td>
<td>resteasy/Resteasy ↔ infinispan/infinispan</td>
</tr>
<tr>
<td>59.</td>
<td>nostra13/Android-Universal-Image-Loader ↔ rgladwell/m2e-android</td>
</tr>
<tr>
<td>60.</td>
<td>nostra13/Android-Universal-Image-Loader ↔ square/wire</td>
</tr>
<tr>
<td>61.</td>
<td>nhaarman/ListViewAnimations ↔ rgladwell/m2e-android</td>
</tr>
<tr>
<td>62.</td>
<td>irstv/H2GIS ↔ irstv/orbisgis</td>
</tr>
<tr>
<td>63.</td>
<td>springside/springside4 ↔ mybatis/mybatis-3</td>
</tr>
<tr>
<td>64.</td>
<td>springside/springside4 ↔ joel-costigliola/assertj-core</td>
</tr>
<tr>
<td>65.</td>
<td>ios-driver/ios-driver ↔ cbeust/testng</td>
</tr>
<tr>
<td>66.</td>
<td>ios-driver/ios-driver ↔ webbit/webbit</td>
</tr>
<tr>
<td>67.</td>
<td>wildfly/wildfly ↔ undertow-io/undertow</td>
</tr>
<tr>
<td>68.</td>
<td>wildfly/wildfly ↔ cbeust/testng</td>
</tr>
<tr>
<td>69.</td>
<td>wildfly/wildfly ↔ weld/core</td>
</tr>
<tr>
<td>70.</td>
<td>wildfly/wildfly ↔ infinispan/infinispan</td>
</tr>
<tr>
<td>71.</td>
<td>wildfly/wildfly ↔ jbosstm/narayana</td>
</tr>
<tr>
<td>72.</td>
<td>wildfly/wildfly ↔ resteasy/Resteasy</td>
</tr>
<tr>
<td>73.</td>
<td>github/android ↔ rgladwell/m2e-android</td>
</tr>
<tr>
<td>74.</td>
<td>square/picasso ↔ rgladwell/m2e-android</td>
</tr>
<tr>
<td>75.</td>
<td>square/picasso ↔ square/wire</td>
</tr>
<tr>
<td>76.</td>
<td>square/picasso ↔ square/okhttp</td>
</tr>
<tr>
<td>77.</td>
<td>alibaba/RocketMQ ↔ alibaba/druid</td>
</tr>
<tr>
<td>78.</td>
<td>alibaba/RocketMQ ↔ netty/netty</td>
</tr>
<tr>
<td>79.</td>
<td>jboss-fuse/fuse ↔ jclouds/jclouds-karaf</td>
</tr>
<tr>
<td>80.</td>
<td>jboss-fuse/fuse ↔ bndtools/bnd</td>
</tr>
</tbody>
</table>
81. jboss-fuse/fuse ↔ wildfly/wildfly
82. jboss-fuse/fuse ↔ fabric8io/fabric8
83. hibernate/hibernate-ogm ↔ infinispan/infinispan
84. hibernate/hibernate-ogm ↔ resteasy/Resteasy
85. DSH105/EchoPet ↔ Bukkit/CraftBukkit
86. DSH105/EchoPet ↔ mbax/VanishNoPacket
87. DSH105/EchoPet ↔ sk89q/worldedit
88. Hidendra/LWC ↔ Bukkit/CraftBukkit
89. elasticsearch/elasticsearch ↔ jboss-fuse/fuse
90. essentials/Essentials ↔ Bukkit/CraftBukkit
91. sk89q/commandhelper ↔ Bukkit/CraftBukkit
92. Multiverse/Multiverse-Portals ↔ sk89q/worldedit
93. Multiverse/Multiverse-Portals ↔ Multiverse/Multiverse-Core
94. richardwilly98/elasticsearch-river-mongodb ↔ cbeust/testing
95. richardwilly98/elasticsearch-river-mongodb ↔ elasticsearch/elasticsearch
96. MinecraftPortCentral/MCPC-Plus-Legacy ↔ SpigotMC/BungeeCord
97. SpigotMC/BungeeCord ↔ jline/jline2
98. SpigotMC/BungeeCord ↔ netty/netty
99. jboss-switchyard/core ↔ jboss-switchyard/release
100. cucumber/cucumber-jvm ↔ hazelcast/hazelcast
101. cucumber/cucumber-jvm ↔ netty/netty
102. cucumber/cucumber-jvm ↔ cbeust/testing
103. cucumber/cucumber-jvm ↔ webbit/webbit
104. cucumber/cucumber-jvm ↔ rgladwell/m2e-android
105. demoiselle/behave ↔ cucumber/cucumber-jvm
106. Multiverse/Multiverse-Core ↔ Bukkit/CraftBukkit
107. capedwarf/capedwarf-blue ↔ undertow-io/undertow
108. capedwarf/capedwarf-blue ↔ infinispan/infinispan
109. capedwarf/capedwarf-blue ↔ resteasy/Resteasy
110. capedwarf/capedwarf-blue ↔ wildfly/wildfly
111. yegor256/s3auth ↔ jcabi/jcabi-github
112. square/wire ↔ square/retrofit
113. square/wire ↔ rgladwell/m2e-android
114. jayway/maven-android-plugin ↔ rgladwell/m2e-android
115. square-okhttp ↔ rgladwell/m2e-android
116. selendroid/selendroid ↔ netty/netty
117. apigee/usergrid-stack ↔ hector-client/hector
118. maxcom/orlsourc ↔ elasticsearch/elasticsearch
119. xXKeyleXx/MyPet ↔ Bukkit/CraftBukkit
120. xXKeyleXx/MyPet ↔ alkarin/BattleArena
121. xXKeyleXx/MyPet ↔ mcMMO-Dev/mcMMO
122. Monstercraft/MonsterIRC ↔ Bukkit/CraftBukkit
123. thinkaurelius/titan ↔ tinkerpop/rexster
124. thinkaurelius/titan ↔ elasticsearch/elasticsearch
125. facebook/presto ↔ jline/jline2
126. facebook/presto ↔ aws/aws-sdk-java
127. facebook/presto ↔ hector-client/hector
128. symphonytool/symphony ↔ overturetool/overture
129. ArcBees/GWTP ↔ gwtquery/gwtquery
130. Jasig/uPortal ↔ Jasig/java-cas-client
131. fabric8io/fabric8 ↔ jclouds/jclouds-karaf
132. fabric8io/fabric8 ↔ wildfly/wildfly
133. fabric8io/fabric8 ↔ jboss-fuse/fuse
134. fabric8io/fabric8 ↔ elasticsearch/elasticsearch
135. taoneill/war ↔ mbax/VanishNoPacket
136. square/dagger ↔ rgladwell/m2e-android
137. square/dagger ↔ square/wire
138. jclouds/jclouds-chef ↔ jclouds/jclouds-karaf
139. sarxos/webcam-capture ↔ netty/netty

The 18 project pairs which have no common users. These are a result of over
approximation of social interactions in SPG.

1. infinispan/infinispan ↔ cbeust/testng
2. mapstruct/mapstruct ↔ weld/core
3. apigee/usergrid-stack ↔ jline/jline2
4. jclouds/jclouds-chef ↔ cbeust/testng
5. jboss-fuse/fuse ↔ jclouds/jclouds-chef
6. fabric8io/fabric8 ↔ jclouds/jclouds-chef
7. fabric8io/fabric8 ↔ bndtools/bnd
8. infinispan/infinispan ↔ jbosstm/narayana
9. capedwarf/capedwarf-blue ↔ fabric8io/fabric8
10. sk89q/commandhelper ↔ jline/jline2
11. MarkehMe/FactionsPlus ↔ sk89q/worldedit
12. xXKeyleXx/MyPet ↔ sk89q/worldedit
13. alkarinv/BattleArena ↔ sk89q/worldedit
14. MinecraftPortCentral/MCPC-Plus-Legacy ↔ jline/jline2
15. xetorthio/jedis ↔ rgladwell/m2e-android
16. Activiti/Activiti ↔ bndtools/bnd
17. Activiti/Activiti ↔ mybatis/mybatis-3
18. dropwizard/dropwizard ↔ joel-costigliola/assertj-core
Appendix C

Timeline Plots for Category A
Project Pairs

For project pairs in Category A, all social activity occurs after the introduction of technical dependence.
Figure C.1: Category A project pairs (1-6)
Figure C.2: Category A project pairs (7-12)
Figure C.3: Category A project pairs (13-18)
Figure C.4: Category A project pairs (19-24)
Figure C.5: Category A project pairs (25-30)
Figure C.6: Category A project pairs (31-36)
Figure C.7: Category A project pairs (37-42)
Figure C.8: Category A project pairs (43-45)
Appendix D

Timeline Plots for Category B Project Pairs

For project pairs in Category B, 75% social activity comes after the introduction of technical dependence.
Figure D.1: Category B project pairs (1-6)
Figure D.2: Category B project pairs (7-12)
Figure D.3: Category B project pairs (13-18)
Figure D.4: Category B project pairs (19-24)
Figure D.5: Category B project pairs (25-30)
Figure D.6: Category B project pairs (31-36)
Figure D.7: Category B project pairs (37-42)
Figure D.8: Category B project pairs (43-48)
Figure D.9: Category B project pair (49)
Appendix E

Timeline Plots for Category C Project Pairs

For project pairs in Category C, less than 75% of social activity occurs after the introduction of technical dependence.
Figure E.1: Category C project pairs (1-6)
Figure E.2: Category C project pairs (7-12)
Figure E.3: Category C project pairs (13-18)
Figure E.4: Category C project pairs (19-24)
Figure E.5: Category C project pairs (25-30)
Figure E.6: Category C project pairs (31-36)
Figure E.7: Category C project pairs (37-42)
Figure E.8: Category C project pairs (43-45)
Appendix F

Communities Detected in UIG

The 22 communities detected in UIG which result in social connections for the 157 project pairs with both technical dependence and social interactions.

In these plots nodes are users and the edges are the interaction between users. Different edge colors within a community represent the different projects over which the user interactions occurred. The captions show the number of vertices or users in the community ($|V|$), the number of edges ($|E|$), and the number of projects on which the interactions took place ($|P|$).

The remaining 82 communities are not shown here. A large number of those communities involve only one project.
(a) $|V| = 288, |E| = 722, |P| = 11$

(b) $|V| = 360, |E| = 768, |P| = 10$

(c) $|V| = 15, |E| = 84, |P| = 2$

(d) $|V| = 527, |E| = 2463, |P| = 21$

(e) $|V| = 263, |E| = 1927, |P| = 8$

(f) $|V| = 16, |E| = 54, |P| = 2$

Figure F.1: Communities detected in UIG (1-6)
(a) $|V| = 50, |E| = 111, |P| = 2$

(b) $|V| = 420, |E| = 1248, |P| = 7$

(c) $|V| = 116, |E| = 141, |P| = 2$

(d) $|V| = 32, |E| = 113, |P| = 5$

(e) $|V| = 692, |E| = 2693, |P| = 23$

(f) $|V| = 671, |E| = 2188, |P| = 8$

Figure F.2: Communities detected in UIG (7-12)
Figure F.3: Communities detected in UIG (13-18)
Figure F.4: Communities detected in UIG (19-22)