10-2012

# Talk versus work: Characteristics of developer collaboration on the Jazz platform

Subhajit DATTA
*Singapore Management University*, subhajitd@smu.edu.sg

Renuka SINDHGATTA

Bikram SENGUPTA

# Talk versus Work: Characteristics of Developer Collaboration on the Jazz Platform

Subhajit Datta
IBM Research, Bangalore, India
subhajit.datta@acm.org

Renuka Sindhgatta
IBM Research, Bangalore, India
renuka.sr@in.ibm.com

Bikram Sengupta
IBM Research, Bangalore, India
bsengupt@in.ibm.com

## Abstract

IBM's Jazz initiative offers a state-of-the-art collaborative development environment (CDE) facilitating developer interactions around interdependent units of work. In this paper, we analyze development data across two versions of a major IBM product developed on the Jazz platform, covering in total 19 months of development activity, including 17,000+ work items and 61,000+ comments made by more than 190 developers in 35 locations. By examining the relation between developer *talk* and *work*, we find evidence that developers maintain a reasonably high level of connectivity with peer developers with whom they share work dependencies, but the span of a developer's communication goes much beyond the known dependencies of his/her work items. Using multiple linear regression models, we find that the number of defects owned by a developer is impacted by the number of other developers (s)he is connected through talk, his/her interpersonal influence in the network of work dependencies, the number of work items (s)he comments on, and the number work items (s)he owns. These effects are maintained even after controlling for workload, role, work dependency, and connection related factors. We discuss the implications of our results for collaborative software development and project governance.

***Categories and Subject Descriptors*** D.2.9 [*Software Engineering*]: Management—Life cycle; H.5.3 [*[Information Systems*]: Group and Organization Interfaces—Collaborative computing, Computer-supported co- operative work

***General Terms*** Experimentation

***Keywords*** Jazz, collaboration, software teams, social network analysis, defects, models.


*OOPSLA'12,* October 19–26, 2012, Tucson, Arizona, USA.


## 1. Introduction

In a collaborative enterprise, association between the structure of communication and the structure of the artefacts produced by the collaboration, has been a rich area of exploration [10]. There is also considerable interest in organizational theory on the role of communication on the performance of teams in knowledge-based work, with the basic premise that effective communication among project team members stimulates the performance of development teams [31]. While new organizational structures have emerged with time, and work practices have been transformed through the use of technology, this central quest has continued to inspire new research. In particular, given the inherently collaborative nature of software development, the domain offers a particularly relevant setting for such investigations, and of late, the influence of "socio-technical congruence" on the performance of software development teams has been gaining attention [8]. In this paper, we report on a real-world socio-technical study of two major releases of a key IBM product developed on the Jazz[1] platform.

Our research has been motivated by two broad lines of inquiry. First, we seek to understand the relation between the "talk" (social) and the "work" (technical) aspects of software development from different perspectives. In our context, the notion of "talk" builds around the *comments* exchanged by developers around units of work – *work items* – and "work" points to the dependencies between the work items owned by the developers. Jazz is a collaborative development environment that supports the definition and tracking of work items and their dependencies, as well as recording developer comments on work items of interest. This, along with the fact that the product development team uses the inherently interactional agile method of development, presented an attractive setting for our study. We abstracted the work item and comment related information from the project into two forms of developer networks: Jazz Collaboration Network based on talk or *JCN-talk* (the "talk network"), and Jazz Collaboration Network based on work or *JCN-work* (the "work network"). We examine and compare several characteristics of these

[1] https://jazz.net/

networks at aggregate and individual levels, to gain useful insights on the socio-technical characteristics of the development team. As a framework for our study, we draw on the network paradigm that has been widely used for studying collaboration in diverse fields [1], [30], [22].

Our second line of inquiry seeks to investigate how the characteristics of developer talk and work can be leveraged towards some practical benefits in project governance. In particular, we hypothesize that a key attribute of a developer's work – the number of defects (s)he owns – will be influenced by a combination of his/her talk and work related attributes. From this perspective, we develop a statistical model to estimate the number of defects a developer owns, based on his/her socio-technical characteristics. Such a predictive model can notably aid expedient resource allocation, realistic time and cost estimation, and several other proactive measures to ensure project success.

Our results along these threads of investigation are organized around two research questions (RQ):

- *RQ-(a): How are the developer characteristics of talk and work related to one another?*
- *RQ-(b): How is the number of defects owned by a developer influenced by his/her talk and work characteristics?*

The results from our study provide a number of interesting insights. Structurally, we found JCN-talk and JCN-work at the aggregate levels to be similar in some aspects – shape of degree distribution, relative size of largest cluster and distance of average separation. The fact that the development team – which is large and distributed – is working towards common project objectives is reflected in some of these similarities. Individually, it was observed that more than half of the developers have a high degree of social connectivity with peer developers they are technically dependent on. However, it was also observed that JCN-talk and JCN-work differ notably in aspects such as density and clustering coefficient, which suggests that a developer's interaction circle through talk typically extends beyond what is indicated by known work dependencies, – in terms of the number of other developers (s)he is connected through talk, and the number of work items that (s)he comments on – was found to depend on a larger span of interest based on his/her team affiliations. We found empirical evidence that a combination of a developer's talk and work characteristics seems to significantly influence the number of defects owned by him/her, after controlling for factors such as workload, role etc. The factors considered in our statistical models for defects is able to account for close to 80% of the variability in the data, and the relatively low mean absolute errors between actual and predicted number of defects makes it a useful prediction mechanism for project governance.

Overall, our study is a valuable addition to the recent and growing body of empirical work that explores the natures of technical and social interactions in software development, and their influence on the performance of individuals.

In the next section we review related literature, followed by an outline of the study settings. Subsequently, we present results and discussion and conclude with a summary of the threats to validity and plans of future work.

## 2. Related Work

### 2.1 Talk and Work in Collaborative Software Development

Conway's observation, "Any organization that designs a system (defined more broadly here than just information systems) will inevitably produce a design whose structure is a copy of the organization's communication structure" [10] – later canonized as "Conway's Law" by Brooks [5] – continues to evoke strong interest in the study of collaborative human enterprise. A perspective on this idea in the context of software development is captured by "socio-technical congruence", which is measured in terms of the difference between coordination requirements due to technical dependencies, and actual coordination [8]. While introducing socio-technical congruence, Cataldo et al. demonstrate that when developers' coordination patterns are congruent with coordination needs during the time-window a modification request (MR) is open, the resolution time of the MR is significantly reduced [8]. This work is extended by Wagstrom et al. to distinguish between communication that is aligned with task dependencies vis-a-vis general communication [33]. They show that the latter has little benefit, while the former reduces bug resolution time. Kwan et al. study the effect of socio-technical congruence on software build success in the IBM Rational Team Concert project [29]. The authors observe that higher congruence actually leads to lower build success rates in certain situations, a large proportion of zero-congruence builds are successful, and socio-technical gaps in successful builds are larger than gaps in failed builds - results which may imply limits to the applicability of socio-technical congruence.

There is a rich body of research in organizational theory that focuses on the impact of collaboration on team performance [6]. Geographically distributed product development continues to be of notable research interest; the significance of informal intra and inter team communication and coordination have been established [7], [23], as has been the need for accumulating a common body of knowledge [12], and for identifying expertise [18]. In our domain of interest – software development – it has long been observed that a team's performance is linked to the effectiveness of collaboration and coordination [28]. The challenges posed by distributed software development in this regard, have been an active area of research [23], [19]. The study of how developer interactions align with the structure of the technical work being carried out, has gained considerable attention of late, inspired

by the socio-technical design theory from organizational literature [26].

However, the relation between talk and work, and how it influences ownership and performance of tasks at the individual level is far from a settled question. While increased communication has been shown to facilitate individuals in some cases [9], too much communication is also detrimental to tasks completion [14], [25]). Communication requirements at the individual level have been recently found to be different than at the team or project level [11]. To investigate these issues in further depth, techniques from social network analysis are being increasingly used to study large scale collaboration [9], [4], [13]. The general direction of such studies has been to examine performance outcomes at the team or project level (e.g. [36], [38]); notable exceptions being [7] and [17]. Bird et al. have studied "socio-technical networks" to predict fault-proneness of software components [3]. In the socio-technical networks of [3], the technical aspect is drawn from inter-component dependencies, and the social aspect is derived from shared contributions to components by developers. In our approach, developer contributions to interdependent units of work are captured within a single work network (JCN-work); while our talk network (JCN-talk) represents a richer model of social collaboration based on co-commenting by developers on work-items of shared interest, which is not considered in [3].

### 2.2 Studies on Jazz Development Data

Over the past couple of years, several studies have been conducted on data from the project developing the Rational Team Concert product using the Jazz platform. This data was made available by IBM for academic research purposes. Some of the initial studies focused on the challenges and opportunities of mining the Jazz platform [24]. Wolf et al. have concluded that distance does not have as strong an effect on distributed communication delay and task completion as was seen in past research [35]. In a subsequent paper, the authors combine a set of communication structure measures into a predictive model that indicates whether an integration build is likely to fail [36]. However, the technical aspects such as interdependencies of the work units are not considered in this study. As mentioned earlier, [29] highlight limitations of the idea of socio-technical congruence. McIntosh et al. study build maintenance efforts of ten projects including Jazz to conclude that build maintenance yields up to a 27% overhead on source code development and a 44% overhead on test development. In a recent paper analyzing the Jazz data, Ehrlich and Cataldo investigated whether centrality and closure help or hinder individual performance [17]. As discussed later, we have included some of the control variables introduced in this paper while building our model. While constructing their "communication networks" the authors of [17] consider the sequence of comments and connect two developers by a directed link only if the latter specifically responded to the former's query or remark. It is not clear from the discussion how comments were matched with responses – any technique requiring human intervention is likely to be subjective and error prone. The authors conclude that individuals who are central within a team's communication network performed better but performed worse when they were central within the communication network for the whole project [17].

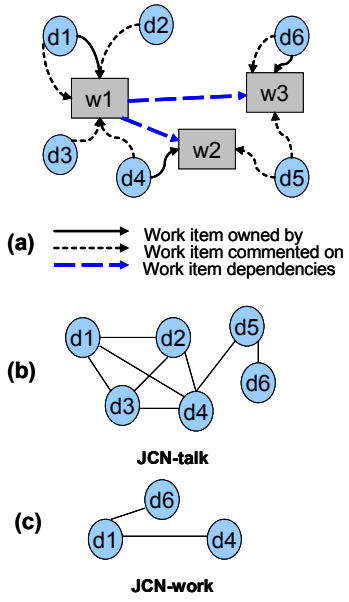


**Figure 1.** Generation of JCN-talk and JCN-work from developer comments, work item ownership and dependencies

All the papers studying collaboration on the Jazz platform so far have been based on the aforementioned data set released for academic research. In contrast, in this paper we study proprietary data from the development of two versions of a major IBM product developed using Jazz.

## 3. Background and Research Setting

We now outline the context of our study in detail.

### 3.1 Collaboration on the Jazz Platform

The Eclipse Way of collaboration which guides Jazz development, defines time bound iteration cycles between two to six weeks [21]. Each cycle has an iteration plan comprising of a set of work items, which are assignable and traceable units of work. Jazz provides predefined types of work items corresponding to different phases of development. Our definitions of JCN-talk and JCN-work are based on the following attributes of work items:

- Work item ownership: Each work item is assigned a developer as the owner. The owner is responsible for the completion of the work item within stipulated time. The

vertices of JCN-talk and JCN-work are such developers who are work item owners.

- Comments around work items: In Jazz development, "*Commenting on the work items is the main task-related communication channel ...* and they provide the context for communication and collaboration." (italics ours) [35]. A developer can post comment(s) on a work item based on his/her interest. Collectively, these comments represent a conversation in a shared context of developer interest around work items. Examples of such conversations are available at the Jazz website.
- Work item dependencies: Jazz supports a set of predefined dependency categories, using which developers can link related work items. This facility has been rigorously used by the development team under study to keep track of work item dependencies. We included the following categories that capture the dependencies most relevant for our analysis: *parent/child* – a work item is broken down into smaller units of tasks; *depends/blocks* – the need for the blocking work item to be completed before the dependent work item can be addressed; *related* - a more generic category that subsumes dependencies based on common features, shared library, code segments etc. In related studies, a common way to define dependencies between units of work has been through code level control/data flow dependencies, or co-changing of the same file [3], [8]. However, our discussions with some senior members of the development team revealed that generic code level dependencies are likely to be too fine grained in our context, and that unrelated work items frequently involve updates to the same file. The "related" category was used to capture relevant code level dependencies when applicable. To avoid many gratuitous connections between unrelated developers, we did not include additional code or co-change based dependency notions in our analysis.

### 3.2 JCN-talk and JCN-work

Figure 1 illustrates the construction of the talk and work networks. Figure 1(a) consists of three work items $w1$, $w2$ and $w3$ with developers owning and posting comments on them. The work items also have dependencies established between them.

- JCN-talk is defined as the network of developers who have commented on work items. With reference to Figure 1(b), each vertex of JCN-talk is a developer and an edge (undirected link) exists between two developers if *both* of them have commented on *at least one* common work item. In the rare instance of the owner of a work item not having commented on it, (s)he is also connected by edges to all others who have commented on the work item. JCN-talk is generated for the set of developers who own at least one work item. In Figure 1(b) edges exist between $d1$, $d2$, $d3$ and $d4$ as $d1$ is the owner of work item $w1$ and $d2$, $d3$, and $d4$ have commented on $w1$; similarly for edges between $d4$ and $d5$ as well as $d5$ and $d6$.
- JCN-work is defined as the network of developers who own work items which have dependencies between them. Each vertex of JCN-work is a developer and an edge exists between two developers if the work items owned by them have a dependency link between them. In Figure 1(c) an edge exists between developer $d1$ and $d6$ as there is a dependency between work items owned by them ($w1$ and $w3$ respectively); similarly for the edge between $d1$ and $d4$.

### 3.3 System Description and Experimental Set-up

As mentioned earlier, we study the internal development data of two consecutive versions of a major IBM product developed on the Jazz platform. Each version is the result of a release that consists of several iterations of development. We refer to this subject of our study as "the product" or "the system" interchangeably, and the versions are denoted by $n$ and $n + 1$. Table 1 presents details of the two versions of the product we studied. All the analysis reported in this paper were replicated for versions $n$ and $n + 1$ of the product. Developer attention on work items often stretch across iterations. For example, a work item may be dependent on another work item planned for an earlier or different iteration, or a work item may be created and discussed before the start of the iteration it is planned for. To capture this natural expanse of work and talk across the development life cycle, we focused on the entire set of iterations leading to a release as our unit of analysis, rather than individual iterations.

The development team of the product is distributed across 35 locations and 19 time-zones across North America, Europe, and Asia. The entire team is divided into smaller subteams or *team areas* – ranging from 3 to 30 members – that focus on specific functionalities of the product. In this paper we are interested in three types of work items – *tasks*, *enhancements*, and *defects* – having source code changes associated with them as "change-sets". Other types of work items that were not associated to source code have not been considered as our study is focussed on development and defect resolution activities. Since we seek to understand the factors influencing the number of defects owned by developers in RQ-(b), based on characteristics of talk and work, we need to ensure a separation of cause and effect while building the models. Thus, only task and enhancement work items are included in the construction of JCN-talk and JCN-work.

We assume that all necessary information for constructing JCN-talk and JCN-work are available in the repository of the Jazz platform. Developers' comments on work items may sometimes be augmented by telephonic or face-to-face conversations. Developers of the project we studied informed us that summaries of such conversations are usually recorded as comments in the Jazz repository. The fact that

**Table 1.** System description

| Parameter | Version $n$ | Version $n+1$ |
|---|---|---|
| No of developers | 107 | 194 |
| No of work items | 6,518 | 11,161 |
| No of work item dependencies | 4,677 | 8,349 |
| No of comments | 20,908 | 40,246 |
| No of iterations of development | 10 | 16 |
| Duration of development | 7 months | 12 months |

significant amount of developer communication is captured in the Jazz repository for the system studied was confirmed by our observation that more than 85% of the developers commented on work items owned by other developers.

To build JCN-talk and JCN-work, relevant information around work items was extracted from the Jazz platform using its client Java APIs. The data was persisted in a specifically designed MySQL database. A set of Java programs was developed to query the database and build the JCN-talk and JCN-work networks in the Pajek[2] file format. The networks were analysed using Pajek, NodeXL[3], and JUNG[4]. Statistical analysis on the data was done using SPSS Statistics 19.

## 4. Results and Discussion

### 4.1 RQ-(a): How are the developer characteristics of talk and work related to one another?

To explore the nuances of talk and work characteristics in depth, we discuss two facets of this question separately.

#### 4.1.1 *Comparison of the structural characteristics of JCN-talk and JCN-work*

As evident from the first row of Table 2, JCN-talk and JCN-work have the same number of vertices representing the *same* set of developers, for each version. The number of edges incident on an individual vertex is referred to as its degree, and across all vertices it is summarized by the *degree distribution*. As evident from rows 3, 4, 5, and 6, the degree distributions for JCN-talk and JCN-work across both versions have positive skews, implying a longer right tail. *This indicates there are relatively few developers connected to many others through talk and work, while many developers are connected to few others.*

The *density* of a network is the ratio of the number of edges actually present and the number of possible edges between a network's vertices. From row 7 of Table 2, we note that JCN-talk is more dense than JCN-work for versions $n$ and $n+1$; their densities vary by more than a factor of 3. *This implies that developers' extents of interest go much beyond the dependencies of their owned work items.*

The largest cluster of a network is the biggest set of vertices all of which are reachable from one another. The relative size of the largest cluster is defined as the percent of the total number of vertices in the largest cluster [30]. In the light of their contrasting densities, it is interesting that sizes of the largest clusters of JCN-talk and JCN-work for both versions are significantly less apart (row 8 of Table 2). Notwithstanding the far less number of edges in JCN-work, more than 75% of developers belong to the largest cluster for both networks. *This indicates that the collaboration channels for talk and work span widely across the team. Within and across different sub-teams, developers are significantly connected to their peers as they work towards common project objectives.*

A key difference between real-world collaboration networks and completely random networks is the phenomenon of *clustering* [30]. It is usually observed in networks of human interaction that two vertices that are linked separately to a third vertex are more likely to be themselves linked. Intuitively, two of one's friends have a higher probability of being friends themselves. This is measured by the *clustering coefficient* of a vertex, which is defined as the ratio of the actual number of edges to the maximum number of edges between the immediate neighbours of the vertex. For the entire network, the clustering coefficient is the average of clustering coefficients across all vertices [1]. The average clustering coefficient is an indication of how likely it is for two vertices to be connected to each other if they are independently connected to a third common vertex. From row 9 of Table 2 we notice the average clustering coefficients of JCN-talk in both versions are in the same range (0.533 and 0.510), and are significantly higher than those for JCN-work (0.249 and 0.189). *These values imply that communities of developers form clusters more readily around discussions on work items, than may be necessitated by the work item dependencies.* This can be explained in the light of a developer's extent of interest being wider than the interdependencies of his/her directly owned work items (as noted earlier and elaborated in subsequent discussion).

The facility for two vertices $i$ and $j$ to contact one another depends on the length of the shortest path $l_{ij}$ between them. The average of $l_{ij}$ over all pair of vertices is called the *average separation* of the network. Average separation is an indication of the interconnectedness of the network [1]. The oft-encountered "small-world" phenomenon with six or lesser degrees of separation are indicated by values of the average separation in networks of human individuals [34]. We note from row 10 of Table 2, that JCN-talk has average separations of 2.194 and 2.305 across versions $n$ and $n+1$, and JCN-work has average separation values of 2.984 and 3.414 across these versions. Thus, over the entire set of 107 developers in version $n$ or 194 developers in version $n+1$, any developer is connected to any other developer over 2 to 4 intermediaries, on average. *The fact that a large team of developers is working closely towards common project*

[2] http://pajek.imfm.si/doku.php

[3] http://nodexl.codeplex.com

[4] http://jung.sourceforge.net/

**Table 2.** JCN-talk and JCN-work: Structural characteristics for version $n$ and version $n+1$ of the system studied

| | Criteria | JCN-talk($n$) | JCN-work($n$) | JCN-talk($n+1$) | JCN-work ($n+1$) |
|---|---|---|---|---|---|
| 1. | Number of vertices ($V$) | 107 | 107 | 194 | 194 |
| 2. | Number of edges ($E$) | 669 | 224 | 1634 | 455 |
| 3. | Mean degree ($AD$) | 12.505 | 4.187 | 16.845 | 4.691 |
| 4. | Median degree ($MD$) | 9.000 | 4.000 | 11.000 | 3.000 |
| 5. | Standard deviation of degree ($SD$) | 10.698 | 4.007 | 17.394 | 5.029 |
| 6. | Skewness of degree distribution ($SK$) | 0.959 | 0.991 | 1.362 | 1.596 |
| 7. | Density ($D$) | 0.118 | 0.039 | 0.087 | 0.024 |
| 8. | Relative size of largest cluster ($L$) | 91.589% | 78.505% | 87.629% | 77.835% |
| 9. | Average clustering coefficient ($C$) | 0.533 | 0.249 | 0.510 | 0.189 |
| 10. | Average separation ($S$) | 2.194 | 2.984 | 2.305 | 3.414 |

*objectives is also reflected in this observation of a small number of intermediaries separating any two developers in JCN-talk and JCN-work, on average.*

After examining the aggregate characteristics of JCN-talk and JCN-work, let us now focus on the individual level.

#### 4.1.2 *Examination of the relation between developer positions in JCN-talk vis-a-vis JCN-work*

We next consider a set of measures that reflect on an individual developer's position in JCN-talk and JCN-work.

In a network, *degree centrality* of a vertex is its degree – the number of edges incident on it [16]. For JCN-talk and JCN-work, a developer's degree indicates the number of other developers (s)he is connected to, over talk and work respectively.

The notion of "betweenness" captures the extent to which a vertex is positioned in the shortest path between other vertices. The *betweenness centrality* of a vertex is defined as the proportion of all geodesics (that is, shortest paths) between pairs of other vertices that include this vertex [16]. When vertices are individuals, betweenness is a measure of interpersonal influence. In our context, betweenness centrality of a developer signifies his/her influence on other developers' channels of collaboration. In this paper, we will use "betweenness centrality" and "betweenness" interchangeably.

The notion of *structural holes* indicates "missing connections and connection redundancies" between vertices [35]. The dyadic constraint on a vertex $u$ exercised by a tie between vertex $u$ and $v$ is given by the "extent to which $u$ has more and stronger ties with neighbours who are strongly connected with vertex $v$" [16]. The aggregate dyadic constraint for a vertex – a structural hole measure – is the sum of all its dyadic constraints. Higher aggregate constraint signifies less freedom to withdraw from existing ties [16]. In our context, developers with higher constraints are more dependent on their existing connections to remain engaged in collaboration. The use of structural hole measures in the study of communication structures in a collaborative context has been demonstrated in [35] and [15].

**Table 3.** Correlation coefficient between developer characteristics ($^*p < 0.01$, $^\dagger p < 0.05$, $^\ddagger p < 0.1$)

| JCN-talk vs JCN-work | V($n$) | V($n+1$) |
|---|---|---|
| Degree centrality | $0.638^*$ | $0.732^*$ |
| Betweenness centrality | $0.392^*$ | $0.667^*$ |
| Aggregate dyadic constraint | $0.176^\ddagger$ | $0.195^*$ |

Degree centrality, betweenness centrality and aggregate dyadic constraint capture different dimensions of an individual developer's position in JCN-talk and JCN-work. How are these positions related between the two networks? Is a developer with high degree centrality in JCN-talk likely to have high degree centrality in JCN-work, and similarly for the other measures?

Table 3 presents the correlation coefficients (along with the $p$ value for statistical significance) between the respective measures for the set of developers in JCN-talk and JCN-work across the two versions. Consistently across versions $n$ and $n+1$, degree centrality has the relatively strongest positive correlation, and aggregate dyadic constraints has the relatively weakest positive correlation (all correlations being significant at the levels indicated) for individual developers across the networks of JCN-talk and JCN-work.

The fact that the degree of JCN-work is found to be strongly correlated to the degree of JCN-talk signifies that *developers with many connections to other developers in the work network also have connections to many collaborators in the talk network and vice-versa.* But is a developer collaborating with the *same* developers via talk as well as work?

To address this question, for each developer we extract the set of developers (s)he is connected to in JCN-talk ($S_{talk}$) and the set of developers (s)he is connected to in JCN-work ($S_{work}$). We define the *Similarity Score* ($SS$) for each developer as, $SS = \frac{|S_{talk}| \cap |S_{work}|}{|S_{work}|}$. Thus $SS$ = 1 for a developer means (s)he talks to *all* the developers (s)he is working with, whereas $SS$ = 0 indicates the developer has *no* common individuals between his/her talk and work collaborators.

**Table 4.** Distribution of Similarity Score

| Bins | Version($n$) | Version($n+1$) |
|---|---|---|
| 0 to 0.25 | 10.714% | 22.086% |
| 0.26 to 0.5 | 16.667% | 11.656% |
| 0.51 to 0.75 | 11.905% | 15.951% |
| 0.76 to 1 | 60.714% | 50.307% |

While computing $SS$, we removed developers who are connected to no other developers in JCN-work (to avoid dividing by zero); there were 23 such developers out of total 107 in version $n$ and 31 out of 194 in version $n+1$. Table 4 gives the frequency distribution of $SS$ as a percentage of the non-zero degree developers of JCN-work. We notice that more than half the developers in both versions are connected to at least 75% (that is, lie in the 0.76 to 1 bin) of their work collaborators in their talk networks. More than 65% are connected to at least half their work collaborators through talk. *These results suggest that the development team members display a reasonably high level of connectivity via talk with those peers they share technical dependencies with*. This augurs well for the overall alignment of talk and work.

In summary, the insights from SQ-(a) are:

- The high value of relative size of largest cluster, and the low value of average separation in the talk and work networks suggest that collaboration links span wide across the distributed team, and the team members are working towards common project objectives.
- The distribution of Similarity Score reveals that developers maintain a reasonably high level of connectivity with peer developers with whom they share work dependencies. At the same time, the marked differences in density and clustering co-efficient across the talk and work networks suggest that the span of a developer's communication goes much beyond the known dependencies of his/her work items.

In the light of these results, let us now address the next research question.

### 4.2 RQ-(b): How is the number of defects owned by a developer influenced by his/her talk and work characteristics?

In the previous research question we examined how the characteristics of talk and work relate to one another at the aggregate and individual levels. Understanding how talk and work characteristics influence a key attribute of individual developers – the number of defects owned – has practical implications for project governance, as well as the potential to enrich the theory of socio-technical congruence.

As with all large scale software projects, significant time and effort in Jazz development is spent on defect resolution. Developers come to own defects that arise in the code modules they were responsible for, and the onus of resolving a defect rests on the developer who owns it. Thus, number of defects owned by a developer reflects an important aspect of their deliverable. Developers owning many defects need to be supported adequately for their timely resolution. The loss of these developers also represent critical risks to the project, so backup plans for knowledge transfer etc need to be in place. Therefore, being able to estimate how many defects a developer will own is of significant value to project stakeholders.

In light of our preceding discussion on the relation between talk and work, we hypothesized that *combination of a developer's talk and work characteristics influences how many defects (s)he owns*. Based on this hypothesis we develop a model using the following parameters.

#### 4.2.1 Dependent variable

As mentioned earlier, every work item on the Jazz platform is assigned to an owner. For each developer the number of defect work items owned by him/her can be determined. We call this variable *NoOfDefects* in our subsequent discussion – this serves as the dependent variable (outcome measure) of our model. Through our model, we seek to identify the parameters that influence this independent variable.

#### 4.2.2 Control variables

Past research has identified several factors that influence individual performance measures in collaborative software development [20], [23], [38]. Based on these results as well as the context specific to our study, the following groups of control variables are used in our model:

- *Control for workload related factor:* As is widely recognized in software development, those who work more get more bugs to fix! Thus, the number of defects a developer owns is impacted by the number of task and enhancement work items (s)he owned in the first place. The variable ***Workload*** – count of tasks and enhancement work items owned by each developer – represents the workload related control factor for our model.
- *Controls for role related factors:* Although agile development encourages a relatively flat hierarchy in the project team, different individuals fulfill different functions in the development ecosystem. In our study, each developer in a team area may have one of the roles – contributor, component lead, tester or member of the project management committee (PMC). A developer may be part of several team areas and it is possible for a developer to have one role in one team area and another role in another team area. As the number of defects owned by a developer is influenced by the specific role (s)he plays in that unit of development, we consider the control variables ***ContributorRole***, ***ComponentLeadRole***, ***TesterRole***, and ***PMCRole***. Few developers who played other roles not covered by the above – such as release management, build coordination – were covered under the ***OtherRoles*** vari-

able. Each of the control variables for role related factors was calculated as the sum of the number of team areas in which a particular developer played that role.

- *Controls for work dependency related factors:* Based on our earlier discussion, aggregate dyadic constraint for a vertex in JCN-work (***WorkConstraint***), measures the level of dependency of a developer on his/her existing connections, and is likely to impact the number of defects (s)he owns. So it is taken as another control factor related to work dependency. In a recent study on the Jazz platform, Ehrlich and Catldo have used "coordination needs" as a control variable, defining it as a measure that "... captures the extent to which an individual needs to communicate with other members of the project given the set of task dependencies that the individual has ..." [17]. Using the method outlined in [17] and [9] we calculated the ***CoordinationNeeds*** control variable for inclusion in the model.
- *Control for connection related factor:* Defects may arise out of connection bottlenecks amongst people who share dependencies with a particular developer. A developer's degree in JCN-work (***WorkDegree***) measures the number of people a particular developer is connected through work dependencies, hence it reflects the extent to which possibilities for such connection bottlenecks exist for that developer. Thus *WorkDegree* represents our connection related control factor.

#### 4.2.3 Independent variables

We now identify the independent variables in our model – these are the talk and work related parameters which we hypothesize will influence the number of defects owned by a developer, after having controlled for the above factors.

- From our discussion around the first research question, we gained insights on how characteristics of talk relate to the characteristics of work. Drawing on those observations, we posit that for a particular developer, the number of other developers (s)he is connected through talk – that is, his/her degree in JCN-talk (***TalkDegree***) – influences a key work attribute – the number of defects owned.
- With reference to our earlier discussion, we recall how betweenness can be construed as a measure of the level of interpersonal influence. Thus, the betweenness of each developer in JCN-work (***WorkBetweenness***) reflects the extent to which (s)he brokers work related dependencies between other developers. As resolution of defects in large scale software systems often involve addressing dependencies across work ownership, we consider *WorkBetweenness* as another independent variable.
- As *TalkDegree* measures the number of other developers a particular developer is connected through talk, the number number of work items (s)he comments on (***NoOfWorkItemsCommentedOn***) indicates how wide his/her span of interest is. As we have seen earlier, it is not unusual for developers to be interested in issues beyond the immediate circle of work items they own. Such general awareness about a project ecosystem is often valuable for defect resolution; thus we take *NoOfWorkItemsCommentedOn* as another independent variable in our model.
- Finally, it is well documented [27] that software defects arise out of dependencies between units of work. Thus the number of defects owned by a developer is likely to be influenced by the number of dependent work items (s)he owns (***NoOfDependentWorkItemsOwned***); we identify this as one more independent variable in our model.

#### 4.2.4 Description of the models

Using the control variables and independent variables described above, we built a set of multiple linear regression models (Table 5). Models I and III are the *base* models using only the control variables, for versions $n$ and $n+1$ respectively. Models II and IV include the independent variables in addition to the control variables for versions $n$ and $n+1$ respectively.

In the models in Table 5, the coefficient of each control and independent variable in the regression equation is mentioned, with its standard error in parentheses below. As noted in the table caption, superscripts of the coefficients denote the range of their respective $p$ values. The $p$ value for each coefficient is calculated using the t-statistic (ratio of each coefficient to its standard error) and the Student's t-distribution. In the lower portion of the table, $N$ denotes the number of data points (in our case, the number of developers in each version), $R^2$ is the coefficient of determination – given by the ratio of the regression sum of squares to the total sum of squares, indicating the goodness of fit of the regression model in terms of the proportion of variability in the data set that is accounted for by the model; $df$ denotes the degrees of freedom; $F$ signifies the Fisher F-statistic – the ratio of the variance in the data explained by the linear model divided by the variance unexplained by the model; and the $p$ value reflects the overall statistical significance of the model, it is calculated using the F-statistic and the F-distribution. For the coefficients as well as the overall regression, if $p <$ *level of significance*, we conclude the corresponding result is statistically significant.

The underlying assumptions of linear regression – *linearity*, *normality*, and *homoscedasticity* of the residuals, and *lack of multicollinearity* between the independent variables – were found to hold within reasonable limits for our analysis [32]. Histogram, P-P plot, and scatter plots of the standardized residuals were used to establish the residual properties. Among the independent variables, only the correlation between *DegreeTalk* and *NoOfWorkItemsCommentedOn* was moderately high (around 0.7) – which is expected, as commenting on many work items makes it more likely for a de-

veloper to be connected to many other developers in the talk network. In light of this, to address whether multicollinearity between variables artificially altered the significance of the overall regression and the stability of the coefficients in our model, the variance inflation factor (VIF) was calculated for each variables. For all the variables, the VIF was found to be less than than recommended limit of 10 (that is, tolerance greater than 0.1) [32]; hence it can be concluded that multicollineraity does not cause significant difficulties for our model.

As the dependent variable is a *count* of defects owned by each developer, Poisson regression was considered as an alternate modeling paradigm. A major threat to the validity of using Poisson regression is *overdispersion*, that is, violation of the strong underlying assumption of equality of variance and mean of a Poisson distribution [2]. We decided to use multiple linear regression in our model as its underlying assumptions were satisfied and it provided notably high goodness of fit, with close to 80% of the variability in the data set being accounted for by the models II and IV (Table 5).

#### 4.2.5 Influence of the independent variables

Comparing the columns of Table 5, we note that for both versions $n$ and $n+1$, addition of the independent variables notably improves the goodness-of-fit over the base models; the $R^2$ value increases by more than 44% between models I and II and by 20% between models III and IV. Thus, controlling for factors related to workload, role, connection, and work dependency, our independent variables *TalkDegree*, *WorkBetweenness*, *NumberOfWorkitemsCommentedOn*, and *NoOfDependentWorkitemsOwned* significantly influence the number of defects owned by a particular developer. To understand the impact of each of the independent variables, let us look at their coefficients in the regression equations.

From the sign of coefficients we can infer the direction of the relationship between the independent and dependent variables. From the columns for model II and IV in Table 5 we note that an increase in *TalkDegree* as well as *WorkBetweenness* for a developer leads to a decrease in number of defects owned, whereas higher *NumberOfWorkitemsCommentedOn* and *NoOfDependentWorkitemsOwned*, leads to higher number of defects being owned. A lower *TalkDegree* implies weaker information sharing channels, and this can negatively impact the quality of work, leading to more defect ownership. Developers who are less "between" other developers in work are likely to find themselves more isolated and hence owning more defects. Interestingly, our model reveals that if a developer comments on more work item (s)he is likely to own more defects. The number of work items commented on is a proxy of the expanse of a developers interests. Anecdotal evidence points to the fact that in large software projects, those who are most knowledgeable about the development context often get the most number of bugs to fix. As a developer who is more knowledgeable about the project will comment on more work items, our empirical findings supports anecdotal evidence. Finally, defects in large software systems often arise out of mishandled dependencies. Thus as our model indicates, a developer owning a large number of work items with dependencies will come to own higher number of defects.

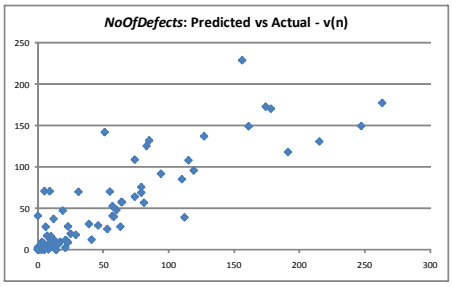


**Figure 2.** Predicted versus actual number of defects for version n

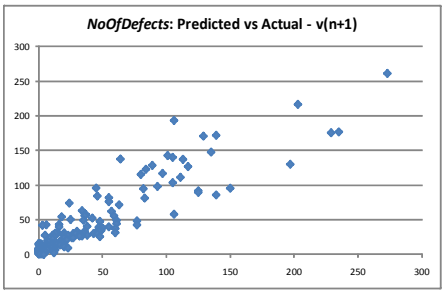


**Figure 3.** Predicted versus actual number of defects for version n+1

#### 4.2.6 Predicting Number of Defects

The models discussed above identifies the factors influencing the number of defects owned by a developer. They can also be used to predict the number of defects, which has strong implications for project governance. In any large scale software project, predicting the number of defects each developer will be responsible for with reasonably high accuracy can be very helpful in resource allocation and quality assurance.

Figure 2 and Figure 3 shows the scatter plots of the predicted versus actual defects for versions n and n+1 respectively. To evaluate the accuracy of prediction of the number of defects calculated using the regression equations based on our models, we consider the *mean absolute error* – the average of the absolute values of the difference between the actual number of defects and the predicted number of de-

**Table 5. Influences of talk and work characteristics on individual performance**. Models I & II are for version $n$ and models III & IV are for version $n+1$ of the product being studied. ($^{*}p < 0.01$, $^{\dagger}p < 0.05$, $^{\ddagger}p < 0.1$)

| | **Model I** | **Model II** | **Model III** | **Model IV** |
|---|---|---|---|---|
| *Intercept* | $-2.107$ | $2.553$ | $-8.472^{\ddagger}$ | $-1.170$ |
| | (9.345) | (6.669) | (5.008) | (4.300) |
| *Workload* | $0.614^{*}$ | $0.265^{*}$ | $0.762^{*}$ | $0.707^{*}$ |
| | (0.189) | (0.150) | (0.091) | (0.079) |
| *ContributorRole* | $-0.753$ | $1.921$ | $8.272^{*}$ | $6.541^{*}$ |
| | (4.060) | (2.800) | (1.980) | (1.610) |
| *ComponentLeadRole* | $-3.392$ | $-5.142$ | $-0.428$ | $-4.499^{\ddagger}$ |
| | (4.918) | (3.493) | (2.934) | (0.088) |
| *PMCRole* | $-6.006$ | $-4.420$ | $-0.4228^{\ddagger}$ | $0.840$ |
| | (3.700) | (2.871) | (2.376) | (2.006) |
| *TesterRole* | $0.246$ | $3.243$ | $0.693$ | $-3.637$ |
| | (21.416) | (14.709) | (6.690) | (5.529) |
| *OtherRoles* | $7.627^{\dagger}$ | $0.776$ | $-2.056^{\ddagger}$ | $-2.074^{\ddagger}$ |
| | (3.631) | (2.571) | (1.336) | (1.240) |
| *WorkDegree* | $5.449^{*}$ | $3.589^{\ddagger}$ | $1.385^{\dagger}$ | $-0.495$ |
| | (1.406) | (2.188) | (0.649) | (0.991) |
| *WorkConstraint* | $-4.547$ | $-2.053$ | $-3.214$ | $-3.089$ |
| | (13.096) | (8.962) | (6.658) | (5.363) |
| *CoordinationNeeds* | $-10.394$ | $-13.028$ | $5.498$ | $3.615$ |
| | (14.572) | (12.067) | (9.023) | (7.784) |
| *TalkDegree* | | $-2.025^{*}$ | | $-0.777^{\dagger}$ |
| | | (0.446) | | (0.197) |
| *WorkBetweenness* | | $-0.055$ | | $-0.022^{\ddagger}$ |
| | | (0.051) | | (0.014) |
| *NoOfWorkItemsCommentedOn* | | $0.643^{*}$ | | $0.288^{*}$ |
| | | (0.061) | | (0.032) |
| *NoOfDependentWorkItemsOwned* | | $0.249^{\ddagger}$ | | $0.241^{\dagger}$ |
| | | (0.161) | | (0.111) |
| $N$ | 107 | 107 | 194 | 194 |
| $R^2$ | 0.553 | 0.801 | 0.645 | 0.775 |
| $df$ | 97 | 93 | 184 | 180 |
| $F$ | 13.348 | 28.747 | 37.201 | 47.756 |
| $p$ | $< 0.001$ | $< 0.001$ | $< 0.001$ | $< 0.001$ |

fects, across all the developers. For the models presented in the columns for models II and IV of Table 5, the mean absolute errors are 16.62 and 12.694 respectively for versions $n$ and $n+1$. In light of the fact that defects per developer range from zero to 263 and from zero to 273 for versions $n$ and $n+1$ respectively, the low values of the mean absolute errors emphasize the predictive utility of the refined models. To check the accuracy of predictions further, we randomly selected 75% of the data points to build the models, and used the models thus built to calculate the number of defects for the test set of the remaining 25% developers. The mean absolute errors for the test set were found to be 20.098 and 13.407 for versions $n$ and $n+1$ respectively; their low values further demonstrating the usefulness of the model as a predictive tool.

### 4.3 Practical Implications of Our Results

The results from our study have practical implications for large, globally distributed software development projects. To begin with, the study underscores the importance of communication in keeping distributed work in synchronization, and highlights the usefulness of collaborative development environments (CDEs) in facilitating the same. Large clusters of individuals with low average separation, high alignment between the talk and work networks of developers, are some of the indicators project managers can look out for as a way to reaffirm that a multi-site team is indeed working cohesively towards common technical objectives. Consequently, deviations from this can act as a red flag that need management attention. The second important implication relates to the overhead associated with communication. The cost of talk is significant in software development, as originally pointed

out by Brooks [5], and which assumes increasing importance in the globally distributed projects of today [37]. While assigned work responsibilities and known dependencies impact a developer's communication overhead, our findings imply that the overall size of a developer's sphere of interest serves as a useful proxy for implicit technical dependencies that may draw a developer's attention and impose additional overload. How effectively relevant information from a developer's ecosystem (both direct and implicit) can be summarized, integrated and presented in CDEs, will determine how well such overload may be managed by development teams of the future. Finally, our findings suggest that a key attribute of a developer's work - as measured by the number of defects owned by him/her - is strongly influenced by a combination of the developer's talk and work characteristics during development. This result is particularly important in light of the high costs associated with testing large software systems. The predictive model we have outlined may be used to identify developers who may be burdened with a high number of defect fixes, and may thus need managerial attention and support. This, in turn, can guide proactive planning and allocation of testing resources.

## 5. Threats to Validity and Future Work

In the following subsections, we identify the limitations of our study with respect to *construct validity*, *internal validity*, *external validity*, and *reliability*.

### 5.1 Construct validity

Construct validity denotes that variables are *measured correctly*. As discussed in the Related Work section, there is a body of existing literature on studying collaboration using the network paradigm, and studies of developer collaboration on the Jazz platform. Almost all the measures used in this paper have been grounded in past work. We have defined the *Similarity Score* metric to examine the level of congruence between a developer's circle of talk vis-a-vis work. As argued during the definition of Similarity Score, we believe this metric succinctly captures the parameter of our interest. In choosing the measures that reflect the control variables and independent variables for the model developed in RQ-(b), we have been guided by past precedent and the specific context of our study. However, we recognize there may be alternative ways of measuring similar variables. The explicit dependency links between work items are manually defined by developers. In few cases there may be implicit dependencies that were not recorded as links and represent another threat to validity. This threat is mitigated by considering other parameters of the developers' sphere of interest, which are likely to capture such implicit dependencies. We have made the links between vertices of JCN-talk and JCN-work non-directional as many of the enduring results of network theory pertain to this class of networks [30]. However directionality of links between developers can provide insights around the nature of information flow in a team. In our future work, we intend to examine the implications of introducing directionality to links in JCN-talk and JCN-work and also consider weights of the links. We also plan to study the effects of assigning different levels of importance to the different categories of dependencies considered while constructing JCN-work.

### 5.2 Internal validity

Internal validity is established for a study if it is *free from systematic errors and biases*. We have accessed development data from the Jazz repository for the two versions of the product studied. As Jazz is the only source of the data we are interested in, issues that can affect internal validity such as mortality (that is, subjects withdrawing from a study during data collection) and maturation (that is, subjects changing their characteristics during the study outside the parameters of the research) do not arise in our case. However the extent to which the development team used Jazz does represent a threat to validity. As mentioned while describing the experimental set-up, our observations as well discussion with the development team suggested that significant amount of collaboration information was captured in the Jazz repository. Thus we believe the extent of this threat to validity is limited. As each developer on the Jazz platform has a unique identifier, our study is free from errors due to ambiguity of individual identities – whether "John Doe" and "J Doe" are the same person or if the same moniker "John C Doe" represent more than one person – that beset other studies on collaboration, such as scientific co-authorship.

### 5.3 External validity

The threat to external validity concerns the *generalisability of the results* of our study. We have studied the association of talk and work in two consecutive versions of a *single* product. As demonstrated in [3], [35], [36], [17] useful insights can be drawn from a single subject of study. The extent of our data – 19 months of development activity, including 17,000+ work items and 61,000+ comments made by more than 190 developers – is notably larger than many similar studies. Thus we believe our results consummately capture the underlying phenomena of interest. But we do not claim these results can be generalized as yet; we plan to study other systems in our future work.

### 5.4 Reliability

Reliability of a study is related to *reproducibility of the results*. As described in the study settings, we employ a set of established techniques to arrive at our results. As there is no human intervention in extracting, and processing the data, there is minimal subjectivity in our results, and they can be reproduced without difficulty.

## 6. Conclusions

In this paper we studied development data from two consecutive versions of a major IBM product developed on the Jazz platform. We analyzed two research questions around the relationship between talk and work characteristics of developers, and how these characteristics influence the number of defect work items owned by a developer. From our results, we observe that characteristics of talk and work reflect the collaboration of a group of developers with individual responsibilities but working towards a common objective, and that the expanse of talk is generally wider than work related dependencies. We developed statistical models to understand the influences of a developer's talk and work characteristics on the number of defects (s)he owns. Standard evaluation criteria established significant goodness-of-fit and high prediction accuracy for the model. Our results highlight some of the subtle and significant aspects of the relationship between developer communication and the units of work around which such communication takes place in a large scale software project.

## References


[1] A. L. Barabasi, H. Jeong, Z. Neda, E. Ravasz, A. Schubert, and T. Vicsek. Evolution of the social network of scientific collaborations. *cond-mat/0104162*, Apr. 2001.

[2] D. Barron. The analysis of count data: Overdispersion and autocorrelation. *Sociological methodology*, 22:179–220, 1992.

[3] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy. Putting it All Together: Using Socio-Technical Networks to Predict Failures. In *Proc. ISSRE 2009*.

[4] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu. Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, SIGSOFT '08/FSE-16, page 2435, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-995-1.

[5] F. P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Addison-Wesley, 1995.

[6] S. L. Brown and K. M. Eisenhardt. Product development: Past research, present findings, and future directions. *The Academy of Management Review*, 20(2):343–378, Apr. 1995. ISSN 0363-7425. doi: 10.2307/258850.

[7] M. Cataldo and J. D. Herbsleb. Communication networks in geographically distributed software development. In *Proceedings of the 2008 ACM conference on Computer supported cooperative work*, CSCW '08, page 579588, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-007-4.

[8] M. Cataldo, J. D. Herbsleb, and K. M. Carley. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *Proc. ESEM 2008*, pages 2–11. ISBN 978-1-59593-971-5.

[9] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, CSCW '06, page 353362, New York, NY, USA, 2006. ACM. ISBN 1-59593-249-6.

[10] M. Conway. How do committees invent? *Datamation Journal*, pages 28–31, April 1968.

[11] J. M. Costa, M. Cataldo, and C. R. de Souza. The scale and evolution of coordination needs in large-scale distributed projects: implications for the future generation of collaborative tools. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, CHI '11, page 31513160, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0228-9.

[12] C. D. Cramton. The mutual knowledge problem and its consequences for dispersed collaboration. *Organization Science*, 12(3):346371, May 2001. ISSN 1526-5455.

[13] K. Crowston and J. Howison. The social structure of free and open source software development. *First Monday, ISSN 1396-0466*, Feb. 2005. The authors examine communication patters of FLOSS projects, finding that FLOSS development teams vary widely in centralizing or decentralizing their communications.

[14] J. N. Cummings. Work groups, structural diversity, and knowledge sharing in a global organization. *Manage. Sci.*, 50(3):352364, Mar. 2004. ISSN 0025-1909.

[15] J. N. Cummings and R. Cross. Structural properties of work groups and their consequences for performance. *Social Networks*, 25(3):197–210, July 2003. ISSN 0378-8733.

[16] W. de Nooy, A. Mrvar, and V. Batagelj. *Exploratory Social Network Analysis with Pajek*. Cambridge University Press, Jan. 2005. ISBN 0521602629.

[17] K. Ehrlich and M. Cataldo. All-for-one and one-for-all?: a multi-level analysis of communication patterns and individual performance in geographically distributed software development. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, CSCW '12, pages 945–954, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1086-4.

[18] K. Ehrlich and K. Chang. Leveraging expertise in global software teams: Going outside boundaries. In *Proceedings of the IEEE international conference on Global Software Engineering*, ICGSE '06, page 149158, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2663-2.

[19] K. Ehrlich, G. Valetto, and M. Helander. Seeing inside: Using social network analysis to understand patterns of collaboration and coordination in global software teams. In *Proc. ICGSE '07*, pages 297–298, 2007. ISBN 0-7695-2920-8.

[20] W. Fong Boh, S. A. Slaughter, and J. A. Espinosa. Learning from experience in software development: A multilevel analysis. *Manage. Sci.*, 53(8):13151331, Aug. 2007. ISSN 0025-1909.

[21] R. Frost. Jazz and the eclipse way of collaboration. *IEEE Softw.*, 24(6):114–117, 2007.

[22] R. Guimera, B. Uzzi, J. Spiro, and L. A. N. Amaral. Team assembly mechanisms determine collaboration network struc-

ture and team performance. *Science (New York, N.Y.)*, 308 (5722):697–702, Apr. 2005. ISSN 1095-9203.

[23] J. D. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *IEEE Trans. Softw. Eng.*, 29:481–494, June 2003. ISSN 0098-5589.

[24] K. Herzig and A. Zeller. Mining the jazz repository: Challenges and opportunities. In *Mining Software Repositories, 2009. MSR '09. 6th IEEE International Working Conference on*, pages 159 –162, May 2009.

[25] P. Hinds and C. McGrath. Structures that work: social structure, work structure and coordination ease in geographically distributed teams. In *Proc. CSCW 2006*, pages 343–352. ACM, 2006. ISBN 1-59593-249-6.

[26] R. Kling, W. Scacchi, and M. C. Yovits. Computing as social action: The social dynamics of computing in complex organizations. volume 19, pages 249–327. Elsevier, 1980. ISBN 0065-2458.

[27] A. G. Koru and H. Liu. Building defect prediction models in practice. *IEEE Softw.*, 22(6):2329, Nov. 2005. ISSN 0740-7459.

[28] R. E. Kraut and L. A. Streeter. Coordination in software development. *Comm. of the ACM*, 38(3):69–81, Mar. 1995. ISSN 00010782.

[29] I. Kwan, A. Schroter, and D. Damian. Does Socio-Technical congruence have an effect on software build success? a study of coordination in a software project. *IEEE Trans. Softw. Eng.*, 37(3):307–324, May 2011. ISSN 0098-5589.

[30] M. E. J. Newman. The structure and function of complex networks. Mar. 2003. SIAM Review 45, 167-256 (2003).

[31] G. L. Stewart and M. R. Barrick. Team structure and performance: Assessing the mediating role of intrateam process and the moderating role of task type. *The Academy of Management Journal*, 43(2):135–148, Apr. 2000. ISSN 0001-4273.

[32] B. Tabachnick and L. Fidell. *Using Multivariate Statistics*. Boston: Pearson Education, 2007.

[33] P. Wagstrom, J. Herbsleb, and K. Carley. Communication, team performance, and the individual: Bridging technical dependencies. Proc. AMC 2010, 2010.

[34] D. Watts. Networks, dynamics, and the Small-World phenomenon. *The American Journal of Sociology*, 105(2):527, 493, 1999.

[35] T. Wolf, T. Nguyen, and D. Damian. Does distance still matter? *Softw. Process*, 13(6):493–510, 2008.

[36] T. Wolf, A. Schroter, D. Damian, and T. Nguyen. Predicting build failures using social network analysis on developer communication. In *Proc. ICSE 2009*, pages 1–11, 2009. ISBN 978-1-4244-3453-4.

[37] N. Zhou, Q. Ma, and K. Ratakonda. Quantitative modeling of communication cost for global service delivery. In *Proc. SCC 2009*, SCC '09, pages 388–395, 2009. ISBN 978-0-7695-3811-2.

[38] T. Zimmermann and N. Nagappan. Predicting defects with program dependencies. In *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, pages 435 –438, Oct. 2009.