Submission: 15720

COMMUNICATION, TEAM PERFORMANCE, AND THE INDIVIDUAL: BRIDGING TECHNICAL DEPENDENCIES

KEYWORDS: SOCIO-TECHNICAL CONGRUENCE, COMMUNICATION, COORDINATION, COLLABORATION, SOFTWARE ENGINEERING, OPEN SOURCE SOFTWARE

ABSTRACT

Many software development projects fail, often one of the major factors is poor communication between individuals on the development team. However, improving coordination in software engineering requires more than just increasing communication among developers. The socio-technical congruence (STC) metric helps to make sense out of complicated the highly-dynamic task dependencies between software developers by analyzing the relationship between developer communication and task dependencies obtained from software artifacts. Previous unreplicated research computed STC from data contained in software repositories and showed that high values of STC in a commercial software engineering setting led to higher team performance, but did not differentiate cleanly between the effects of aligned communication and the effects of general communication. In this work we utilize software repository data from an Open Source software community to evaluate STC in a very different setting. We then extend STC, creating an individual metric and two sub-measures that distinguish between communication that is aligned with task dependencies and general communication, and show that general communication has little benefit, while communication aligned with our measure of task dependencies reduces bug resolution time.

INTRODUCTION

"If we could only get developers to talk to one another we would never have most of these problems." Such a hypothetical lament would not be unexpected in almost any software development team. As the market continues to evolve, software is frequently developed by dynamic and virtual teams. Frequently forming for short term tasks (Townsend, Demarie, & Hendrickson, 1998), dynamically adding and removing team members (Lipnack & Stamps, 2000), and often incorporating members at remote sites (Hinds & Mcgrath, 2006), communication amongst members is never a given and is rarely predictable. To be successful, teams must contend with gaps in organizational knowledge (Becker, 2001; Druskat & Kayes, 2000), differing work and social norms (Jarvenpaa & Leidner, 1999; Montoya-Weiss, Massey, & Song, 2001), a lack of experience working together (Cummings & Kiesler, 2008) and transactive memory (Ren, Carley, & Argote, 2006), varying skill sets (Jackson, 1999), and the structure of the social network of work relations (Hinds, Carley, Krackhardt, & Wholey, 2000) .

As teams grow in size, subteams often form around individual components within the system. For example, a software development team working on building a new web browser may have different subteams centered around the major components of the project: user interface, network communications, page rendering, and security. Such structuration of the team often results in the structure of the final project mirroring the structure of the team designing and implementing it, as was first noted by Conway (Conway, 1968). While team members are often acutely aware of with whom they directly collaborate on tasks and components, the reality is that tasks can rarely be considered in isolation. Most components of complicated technologies have dependencies on other components, some of which may be owned by the same team, some of which may owned by other teams. These dependencies across components may not be immediately obvious to team members, and may require additional communication beyond what is done through standard communication patterns (Sosa, Eppinger, & Rowles, 2004). As teams become more complex, introducing more team members across other sites, the communication and coordination dependencies increase (Cataldo & Herbsleb, 2008). In particular, as workload becomes asymmetric between subteams and sites, identification of key individuals at remote locations becomes increasingly difficult (Bayerl & Lauche, 2008; Grinter, Herbsleb, & Perry, 1999). The provisioning of a tool that addresses these needs has shown potential to dramatically improve team performance (Merrill et al., 2008).

Such a premise forms the basis for the Socio-Technical Congruence metric (Cataldo, Herbsleb, & Carley, 2008; Cataldo, Wagstrom, Herbsleb, & Carley, 2006). Briefly stated, Socio-Technical Congruence (STC) relates the actual communication of an organization to the coordination requirements of that organization as derived from the dependencies among the tasks performed. From a mathematical perspective, Socio-Technical Congruence views organizations and tasks as networks that embody relationships amongst individuals and individuals, tasks and tasks, and between individuals and tasks. Using network analytic methods (Carley & Krackhardt, 1998) it is possible to combine and manipulate these networks to create new networks that shed light on non-obvious relationships within the organization.

For our analysis in this paper we formulate these network relationships using matrix algebra, although it is equally possible to utilize a graph theoretic formulation (Valetto et al., 2007). Formally, in matrix notation one needs a network of individuals and tasks, called the task assignment matrix, $\boldsymbol{T_A}$, which maps individuals to tasks. The cell i, j in $\boldsymbol{T_A}$ is one if individual i

worked on task j, and zero otherwise. In addition to a task assignment network, another network relating tasks to one another, $\boldsymbol{T_D}$, the task dependency network is required. This network has a one in cell i, j if task i is dependent on task j and zero otherwise. Using these two networks it is possible to calculate the degree to which two individuals are dependent on one another through their tasks through a simple matrix multiplication:

$$\boldsymbol{C_R} = \boldsymbol{T_A} \times \boldsymbol{T_D} \times \boldsymbol{T_A}'$$

This derived network, $\boldsymbol{C_R}$, is called the coordination requirements network. To avoid issues with a very non-uniform distribution, address possible repercussions of noise in the data, and because preliminary evaluations have shown little to no benefit from the increased complexity of valued networks in the framework of STC, $\boldsymbol{C_R}$ is typically dichotomized to a binary network and has a one in cell i, j if individual i and individual j are working on tasks that are dependent on each other and a zero if they are not. Because these coordination dependencies arise from the dependencies between tasks and not necessarily from team structure they are often not apparent to the individuals working on a project. For example, if Alice is developing the page renderer for a web browser and Bob is developing the network interface, and the page renderer requires data from the network interface, then Alice and Bob have a coordination dependency even though they do not directly collaborate on the same component. Frequently these networks can be obtained through automated methods, such as inspection of formal roles in an organization or analysis of artifacts as a result of the design and implementation process (Lopez-Fernandez, Robles, & Gonzalez-Barahona, 2004). In most cases, networks used for calculating Socio-Technical Congruence are dichotomized after each step to simple binary matrices.

The final component for evaluating Socio-Technical Congruence in a team is a network of the actual communication in an organization, $\boldsymbol{C_A}$. Within this network, cell i, j is one if individual i communicated with individual j and zero otherwise. We say that if i, j is one in both $\boldsymbol{C_A}$ and $\boldsymbol{C_R}$ that there is an instance of matched communication. The overall Socio-Technical Congruence for the organization is then the ratio of matched communication to the total number of coordination requirements and is bounded between zero and one. Formally, with binary networks this metric can be specified using the logical conjunction as:

$$\boldsymbol{STC} = \frac{\sum(\boldsymbol{C_A} \wedge \boldsymbol{C_R})}{\sum \boldsymbol{C_R}}$$

Previous research by Cataldo et. al. examined the relationship between Socio-Technical Congruence and the time to resolve tasks in a commercial software development environment (Cataldo et al., 2006). It was found that after accounting for common factors such as team size, experience, and task complexity, when the team had higher levels of Socio-Technical Congruence it was able to complete tasks faster. This research also evaluated the time-based evolution of Socio-Technical Congruence within a team and found that as the team evolved the Socio-Technical Congruence increased, indicating that the team was doing a better job of understanding dependencies and communicating across those dependencies. Yet it was only the top 10% or so of developers, as measured by productivity, who consistently communicated in highly congruent ways. Although the other 90% improved somewhat over time, they never learned to use communication channels nearly as congruently as the best developers.

This work of Cataldo et. al. evaluated the communication and performance of the team at the organizational level. From a managerial perspective, this is helpful as it provides a single number that can be used to help gauge the health of the organization (Cataldo et al., 2006). However, from an individual perspective this metric may be problematic. Firstly, the network based nature of the metric makes it difficult for an individual to understand how well they personally are communicating across coordination dependencies. Secondly, the metric is constructed in such a way that it is not easy to differentiate between individuals who communicate frequently across all possible links, and those that communicate primarily across coordination requirements. Thus, in the extreme case an individual who continually made contact with everyone on their team would have perfect STC, even though there is little chance of actually satisfying all of the coordination dependencies.

This ambiguity reveals a significant limitation of the previous study. It is known that in many fields, highly central individuals (e.g., those with communication links to many other people) also tend to be high performers, and this is true in software teams as well (Cataldo & Herbsleb, 2008). Since highly central people communicate with many others, they are more likely to have communication that matches coordination requirements, just by chance. It is therefore not possible to say, based on the organization-level STC measures, whether

a) congruent communication improves performance (the STC hypothesis) or
b) top performers tend to be central in communication networks, and therefore have more communication links of all types, including congruent communication (the centrality hypothesis).

This paper addresses these issues by first proposing a set of extensions to the Socio-Technical Congruence metric. First, to make the metric more coherent and applicable to individual behaviors, we extend it beyond an organizational level metric to an individual level metric. It then addresses the utilization of communication that matches coordination dependencies and communication that is unmatched with regards to task dependencies to evaluate the individual effectiveness of the metric. This allows a differentiation between individuals who are highly central in a network because they communicate frequently across all channels and those individuals who focus more on coordination dependencies. We then replicate the original finding regarding Socio-Technical Congruence on a large Open Source Software community, the GNOME project. This is followed up with an evaluation of the individualized Socio-Technical Congruence formulation on the same data and we find an even greater benefit to the organization. Next we evaluate the impact of communication that matches coordination dependencies and communication that is unmatched with regards to coordination dependencies, finding that communication that matches these dependencies has a much greater beneficial impact on the project than communication that does not match such dependencies, strongly confirming the STC hypothesis and providing no support for the centrality hypothesis. Finally, we close with a discussion of the implications of this metric for organizational design and the creation of tools to help foster organizational communication and coordination.

INDIVIDUALIZED SOCIO-TECHNICAL CONGRUENCE

The limitations of the original STC metric first became evident as efforts were undertaken to develop tools that automatically calculated STC. It became clear that the network nature of the original STC metric made it difficult for individual developers to understand how their individual actions contributed to the overall network level STC. This resulted in the creation of the individualized STC metric, the formula for which can be seen below:

$$ISTC_i = \frac{\sum(\boldsymbol{C_R}[i,] \wedge \boldsymbol{C_A}[i,]) + \sum(\boldsymbol{C_R}[,i] \wedge \boldsymbol{C_A}[,i])}{\sum \boldsymbol{C_R}[i,] + \sum \boldsymbol{C_R}[,i]}$$

Like the original network level metric, individualized STC utilizes binary networks that have been dichotomized from valued data. The individualized STC, denoted as ISTC, for individual i is calculated by examining only the elements of the matrix that are incident upon individual i. The numerator calculates the number of coordination requirements for individual i that have matching communication over both the rows and columns for that individual. The denominator simply sums up the number of coordination requirements incident upon individual i. ISTC provides a metric for every individual in an organization and can be used to more easily identify points where communication is lacking. When taken over the entire organization, the mean ISTC for each of the individuals is the overall STC for the organization.

Another issue that arose in the development of STC related tools and that we seek to correct in this work arises from the fact that STC is formulated as a fraction of communication that was matched with coordination requirements divided by the number of coordination requirements. If used as a team assessment tool this results in individuals and projects with few coordination requirements exhibiting high levels of STC that are impossible for individuals with more responsibility and complex projects to obtain. We propose providing another method to analyze STC by splitting the fraction into its constituent parts: matched communication and coordination requirements.

$$STC = \frac{\sum(\boldsymbol{C_A} \wedge \boldsymbol{C_R})}{\sum \boldsymbol{C_R}} = \frac{matchComm}{coordReq}$$

By breaking apart the STC fraction, this allows evaluation of the scale of projects and also allows the differentiation between the effects of high and low coordination requirements and matched communication. A third element that this formulation provides is the ability to integrate the "extra" communication into our analysis, where extra communication is defined as communication present in the actual communication network, $\boldsymbol{C_A}$, for which there is no corresponding edge in the coordination requirements network, $\boldsymbol{C_R}$. Formally then, we define $matchComm$, $coordReq$, and $extraComm$ for each individual, i, as follows:

$$matchComm_i = \sum(\boldsymbol{C_R}[i,]\wedge\boldsymbol{C_A}[i,]) + \sum(\boldsymbol{C_R}[,i]\wedge\boldsymbol{C_A}[,i])$$

$$coordReq_i = \sum \boldsymbol{C_R}[i,] + \sum \boldsymbol{C_R}[,i]$$

$$extraComm_i = \sum \boldsymbol{C_A}[i,] + \sum \boldsymbol{C_A}[,i] - matchComm_i$$

In addition to addressing issues with large scale projects the use of $matchComm$ and $extraComm$ (which sum together to all of the communication) allows the differentiation between the effects of merely communicating with others in a team, and communication across dependencies to address coordination needs within the team.

DATA COLLECTION

Calculation of STC and ISTC requires three major pieces of information: the task assignment network, $\boldsymbol{T_A}$, that maps individuals to tasks, the task dependency network, $\boldsymbol{T_D}$, that shows dependencies between tasks, and a network of actual communication, $\boldsymbol{C_A}$, that shows the communication between individuals in the community. In addition, to test for significance of the metric it is necessary to collect an outcome metric related to performance.

A natural source for such data is an examination of teams that almost exclusively utilize computer mediated communication. If the tools used by the team collect and archive all of the communication and actions by team members it is a simple matter of mining the historical archives. Open Source Software development, which sees individuals (both volunteers and professionally paid) work together using very lean tools to develop high quality software is a natural source of such data (Halloran & Scherlis, 2002; Mockus, Fielding, & Herbsleb, 2002). Because these communities are often voluntary in nature and span the globe, most projects use a very minimal set of tools that are easily accessible to all project members. Most communication and coordination takes place over project mailing lists, the software is developed and committed back to a common version control system that records modifications made by each developer, and project management and bug tracking is typically done via web based systems such as Bugzilla (Fogel, 2005; Yamauchi, Yokozawa, Shinohara, & Ishida, 2000).

The primary source of information for task assignments and task dependencies was the version control system (VCS). When using most VCS systems, each software developer downloads a complete copy of the project source code to his/her own computer, modifies a set of files, and commits them back to the main VCS server. The VCS not only records that the developer made modifications to the files, but which files were changed together and metadata about the modification, such as a short log message and the time of the modification. In most software engineering projects the project source code is broken in to multiple files or modules, where each file or module implements a feature or set of related features. For our purposes, we consider a single file to be a task that may be worked on by multiple individuals. Thus, if the VCS has a commit from developer i on file j then a one is placed in cell i, j of the task assignment matrix, $\boldsymbol{T_A}$, and a zero otherwise.

Because the VCS records information about all changes to files made in a commit, it is possible to obtain information about combinations of files that are committed together in a single commit action. The norms of the Open Source development process encourage developers to make small, incremental commits back to the VCS (Hertel, Niedner, & Hermann, 2003). These commits contain files that are related to each other, for example all of the files required to implement the feature of adding a signature line to outgoing messages in an email client. We infer a dependency between files that were committed together, a method called logical dependencies, which is programming language agnostic and has previously been shown to accurately reflect the real dependencies in a software engineering context (Gall, Hajek, & Jazayeri, 1998). Using this metric, if file j and k were both committed to the VCS in the same commit, a one would be placed in cell j, k of the task dependency matrix, $\boldsymbol{T_D}$. We chose logical coupling over other methods such as static or dependency analysis because it works for programs written in any programming language and is consistent thanks to its clear formulation, whereas static and dependency analysis tools are often language specific and may have differing degrees of coverage.

Finally, most mailing list software provides the ability to archive all messages – a feature designed to assist users in searching for previously discussed information, but also one that allows the reconstruction of complete conversation networks. In addition, the norms of many communities, which dictate that discussion and decisions should be made on project mailing lists, help guarantee that most of the communication between team members is captured in this medium (Ducheneaut, 2005). There exists multiple methods to generate communication networks from mailing list archives (Bird, Gourley, Devanbu, Gertz, & Swaminathan, 2006), we adopt the convention that links are created in the actual communication matrix, $\boldsymbol{C_A}$, between person i and j if person i does one of the following: explicitly lists person j as a recipient of the message in the "to" or "cc" fields, responds directly to a message j as determined by the message headers, or posts a message in a conversation thread which j has already posted a message to.

One notable advantage in collection of data using these methods is that all data is automatically collected and made available through automated tools. Therefore, the collection and use of these data streams for the purposes of calculating STC and ISTC can be completely automated and easily integrated into collaboration and awareness tools.

Study Sample

The specific community of interest was the GNOME project, a large and successful Open Source community dedicated to building a complete desktop environment for Linux and UNIX systems. The project was founded in 1997 and has seen significant growth and corporate involvement from many large and successful IT firms, such as Red Hat, Novell, and IBM (de Icaza, 2002; The GNOME Foundation, 2008). Although GNOME still retains substantial elements of its informal volunteer roots, the project has matured and now has substantial processes in place for integrating new members, managing contributions, and planning software releases. In many cases these processes are very similar to those that would be found in commercial environments, making the software development process within this largely volunteer community more similar to commercial software firms than one may initially assume (Koch & Schneider, 2002; German, 2004).

The community around the GNOME project operates in a federated manner wherein the larger project is responsible for planning major releases and community wide issues and each sub-project within the project operates relatively independently, approving new developers, planning future development, and managing bugs on their own. Despite the independence of subprojects to decide their own work practices, the overlap of developers between many of the projects in the community and the strong sense of community norms in the broader Open Source community results in very similar management practices across most projects.

There are hundreds of different sub-projects that make up GNOME, but many of these projects have only small amounts of developer participation or have started outside of the GNOME project and still use external services to provide critical pieces of infrastructure, and therefore complete archives of communication and source code may not be available. To provide a robust analysis, only sub-projects with more than 20 developers, 100 bugs in the bug tracking system, at least one publicly archived mailing list, and more than a year of overlap between mailing lists, source code history, and bug tracker data were used in the analysis. This yielded fourteen projects.

The community also follows several norms which made generation of networks, particularly the task dependency network, $\boldsymbol{T_D}$, difficult. There were a number of files that were frequently committed and had little to do with implementation of functionality, in particular project documentation, compilation scripts, and the `ChangeLog` file that documents all changes made to the project source code. To avoid generating an overly dense network that does not reflect the actual structure of tasks and dependencies within the project, only source code files were taken into account when generating the task dependency and task assignment networks. Second, there were a number of commits that touched a large number of files, sometimes numbering into the hundreds. Examination of such commits revealed that these commits were typically custodial in nature, such as updating headers in the source code, changing the formatting of the source code, and restructuring the directory layout for the project. To avoid such instances, commits that touched more than 20 files were removed from the data set. This value was chosen based on expertise in the community and an examination of fifty randomly selected commits that touched 20 or more files – it was found that only three of those commits added functionality, and they were a combination of a series of patches contributed by users without direct access to the repository.

The final source of data obtained from the community was a complete copy of the project bug database, Bugzilla. In addition to recording simple bug reports, Bugzilla is a complete defect management system that allows users to start discussions around bugs, post attachments for bugs (e.g. test cases, proposed patches), and also change the status of a bug (e.g. from new to fixed). Each change in the system is recorded to create a complete provenance trail for each bug. We used data from the Bugzilla database to get the set of bugs for each of the sub-projects and the length of time that each of the bugs was open – the amount of time elapsed from when it was first entered into Bugzilla to the final time it was marked as resolved in the database.

Most of this data was collected automatically – either through publically accessible archives or by working with project system administrators to obtain a copy of the data from the main project servers. One notable disadvantage to evaluating this community is that there was no direct integration between the systems. For each of the projects the identities of the users were unified across all three platforms by hand and verified by members of the community. This was the only part of the data collection and analysis that could not be accomplished with automated methods and is largely an artifact of the history of the community, which was founded before integrated tools and environments were available to Open Source teams.

DATA ANALYSIS/EVALUATION

Replication of STC at a Network Level

To verify the findings of Cataldo et. al. (Cataldo et al., 2006) on an automatically collected data set and to ensure that STC had a similar effect in a largely volunteer Open Source setting, a regression model was created to predict the time to resolve bugs in the software based on the overall STC of the sub-project in the time window that the bug was open (defined as the number of days from when bug was first reported until the final time the bug was marked as resolved by project developers). This is the same dependent variable used in the original analysis. To avoid issues with trivial bugs and enhancements that sometimes are reported to project bug trackers, we filtered the data so only bugs that were open for more than one day were used. Another problem was that the duration of bugs was highly skewed, with most bugs remaining open for under two months, but a handful of bugs remained open for years. To more closely approximate a normal distribution, we followed the common practices of using a $\log_2$ transform of the time the bug was open as the dependent variable in our model.

Beyond the level of STC present in a project, there are numerous factors that impact the amount of time it may take to resolve software defects. Among them are personal traits; such as years of experience programming, education attained, and formal role within the team; and bug traits, such as the perceived priority of the bug, the number of components the bug involves, and even who submits the bug in the case of large customers who receive prioritized service (Brooks, 1995; Curtis et al., 1986). As one of our goals with this work was to replicate the findings of Cataldo et. al. (Cataldo et al., 2006) using automatically collected data in an Open Source environment, we have attempted to use control variables similar to those in the original work, however, while it may be possible to obtain many of these variables in a commercial organization, the volunteer nature of Open Source makes it difficult to match all of those control variables. By analyzing the project history we were able to find a number of control variables that are typical for understanding software engineering processes. The control variables are as follows:

numDevs – The number of developers active on the bug report. A developer was considered to be someone who had directly committed code to the project CVS repository. It is expected, as Brooks noted, that more developers working on a bug will result in longer resolution times (Brooks, 1995).

numDeltas – The number of changes to the bug status during the period the bug was open. For example, changing a bug from "NEW" to "ASSIGNED" is a single delta. Bugs that have high numbers of status changes may be indicative of difficult or contentious bugs that are frequently reassigned.

numComments – The total number of comments on the bug. Every bug report has at least one comment describing the original bug, but most have multiple comments that can be followed to trace the workflow in understanding the bug. Many comments on a bug may indicate particularly difficult bugs.

severity – Each bug is assigned a severity in the bug tracker that ranges from trivial to critical. The seven levels were coded from 0 (lowest priority) to 6 (highest priority).

projectAge – the $\log_2$ of the number of days that project had been active at the time the bug report was filed. The starting date for a project was determined by the earliest date of the first bug report, mailing list message, or commit to project source code. This variable is a proxy for changes in a project over time, for example the number of people may grow or shrink, or the code base may become larger and more difficult to change.

numBugs – The number of new bugs filed on the project in the period immediately surrounding this report (± 2 months). This variable is a proxy for variations in the overall project workload.

devExperience – the $\log_2$ of the number of days since the developer's first activity in the broad community. Developers who have been active longer should have additional experience and therefore be able to resolve software defects faster. In the case of multiple developers active on a single bug, this value is the average across those developers.

cvsCommits – A binary variable indicating whether or not the bug has associated commits in the CVS source code repository. As there is no direct integration between the bug tracker software and CVS software, this was found by examining if there was a CVS commit with a log message that referenced the bug report. This sort of enhanced tracking may be an indicator of more formalized process and management within a project.

The correlations of the control variables are shown in Table 1. We note that while most of the control variables have low correlations, there exists a collection of variables related to bug activity that exhibit higher correlation. We address the implications of these correlations later in this paper.

Insert Table 1 about here

The control variables and the overall STC of the organization at the time the bug was filed were then used to create a linear regression model to estimate the $\log_2$ of the time to resolve bugs in projects within the ecosystem, we call this base regression model 1.

The regression was run with 26512 non-enhancement related bug reports from projects that were filed on the previous identified subprojects in the GNOME ecosystem. The results are shown in the first results of . As has been shown in previous research, the more developers, $numDevs$, that are active on a bug, the longer it will take to resolve (Herbsleb, Mockus, Finholt, & Grinter, 2001). Unexpectedly, an increase in the number of status changes, $numDeltas$, decreases the time to resolve the bug. The more conversation on a bug report, $numComment$s, reduced resolution time, as did a high severity for the bug. There is evidence that more mature projects, both in terms of overall age, developer experience, and process ($cvsCommits$) were able to resolve defects faster. Bugs that were filed during a period with a high number of other bugs filed, $newBugs$, were found to take significantly longer to resolve. Finally, we examine the effect of organizational STC for each bug. The value used for each bug was the project-wide STC as of the day that the bug was first reported, therefore two bugs that were reported on the same day will have the same value of STC. This is in line with the original findings regarding STC from a proprietary software development firm (Cataldo et al., 2006), we find that higher levels of STC in the organization lead to a decrease in the time to resolve bugs. The fit of the model is very good and produces fairly accurate estimates for most bugs – however it has some difficulty estimating the time of very short or very long lasting bugs.

Insert Table 2 about here

Examination of these very short and very long bugs yielded no obvious information that may result in their deviations from predicted resolution times. We therefore believe that these bugs have additional external factors that are not present in the dataset and would be otherwise difficult to automatically obtain for the data. Nonetheless, we consider this model to be highly successful because of three major reasons: 1) it is an empirically sound replication of the effect STC 2) the effect of STC was found to be significant even in the radically different workspace of Open Source software development, where developers often come and go more frequently and have some choice in their tasks, and 3) the entirety of the data set for the analysis was automatically collected with no human-in-the-loop.

Insert Figure 1 about here

Individualized STC

Having established the relationship between higher levels of STC at the project level and a lower time to resolve bugs, the next step was to evaluate the relationship between an individual's communication patterns with respect to coordination requirements and the time to resolve software defects. As before, the unit of analysis is the bug report, the dependent variable remains $\log_2$ of time to resolve the bug, and the control variables remain the same as in model 1. However, in contrast to the previous section, rather than using overall organizational STC during the time period the bug was closed, we now use the ISTC of the developers active on the bug. For a majority of bugs, 63%, only a single developer was active on the bug (however, numerous other non-developers may have been active on the bug), on the 37% of bugs for which more than a single developer was active on the bug, the mean value of ISTC across those developers was used. The distribution of values for ISTC can be seen in Figure 1.

A similar regression model to the previous regression model was used, with the exception of replacing STC with ISTC, the average individual STC for the developers involved on each bug. Within this new model, dubbed model 2, the distribution of ISTC is near normal, as shown in Figure 1. The evaluated coefficients for the model are shown in the middle column of Table 2.

We find that the control variables remain at substantially similar levels to their levels in model 1, indicating stability in the model. However, in the new model that incorporates only the ability of developers directly involved with the bug to address coordination requirements, ISTC is has a coefficient that is more than 50% greater than the coefficient for STC found in model 1. This model indicates that a bug which is assigned to a developer with a very high level of ISTC may be completed approximately 16% faster than a task that is assigned to a developer with a very low level of ISTC.

Differentiating Communication and Coordination Requirements

While these first two models provide encouraging results regarding the validity of STC and ISTC, the data we have allows the examination of the relationship between individual developer communication, technical dependencies, and actual communication and coordination at a level never before possible. This allows us to address whether or not high levels performance is due to developers who know how to properly communicate across coordination dependencies, or whether these developers with the highest performance are merely highly central in the network and communicate without regard to coordination requirements. Here we construct a new model, model 3, that has the same dependent variable and control variables as the two previous models. Rather than using STC or ISTC as the variables of most interest, this model breaks apart the fractional nature of ISTC and uses the raw counts of coordination requirements ($coordReq$), communication that is aligned with coordination requirements ($matchComm$), and extra communication that exists but has no corresponding coordination requirement ($extraComm$) – a factor previously not accounted for in models 1 and 2. Instances where there is a coordination requirement between two individuals but where there is not communication, also known as gaps(Valetto et al., 2007), are not expressly represented as they are a linear combination of existing factors ($coordReq$ and $matchComm$) and thus their inclusion would create a singularity in the model. If it is the case that being highly central in the network of developers leads to increased performance then there should be little difference between the effects of $matchComm$ and $extraComm$. The results of the model can be seen in the right hand section of

The various control variables are very similar to those found in both model 1 and model 2, indicating model stability. As expected, an increase in the raw number of coordination requirements leads to an increase in the amount of time to resolve software defects. This is in-line with previous observations that work often proceeds slower in technically complex projects (Brooks, 1995). However, for each time that a developer communicates along the lines of a coordination requirement, this increase in time to resolve the bug is more than negated. By controlling for any extra communication between developers, which has a coefficient that is not statistically different from zero at the p=0.05 level, we have been able to differentiate between the impact of communication that addresses technical dependencies and the communication that does not.

Insert Figure 2 about here

A further analysis of the actual vs. fitted times to resolve bugs, as shown in shows that the model tends to have difficulty with the wide range of bugs that are closed almost immediately after opening. However, for a majority of bugs, the model is fairly accurate. The accuracy of the model trails off for those bugs with extremely long resolution times – in particular those bugs that take two or more months to resolve tend to be under-predicated by the regression model.

As a final step, the standardized β and variance inflation factors were calculated for each of the factors in Model 3 and are shown in Table 3. The calculation of standardized β allows the ranking of the factors in the model in terms of the variance explained. The variance inflation factors can be used to address whether multicollinearity between factors artificially altered the significance of the overall regression and the stability of the coefficients. The standardized β values show that the largest factor in predicting the time to resolve a new software defect is the number of defects that the team is the team workload around the time that the bug was filed, $newBugs$. This is followed by the coordination requirements of the team, $coordReq$, the number of changes made to the bug's status, $numDeltas$, experience of the developers assigned to the bug, $devExperience$, and finally the amount of communication that matches coordination requirements, $matchComm$.

Insert Table 3 about here

The analysis of the standardized β scores indicates that although there was high correlation between the trio of $numDevs$, $numDeltas$, and $numComments$, and again between $numDeltas$ and $newBugs$, there was little increase in the variance in the model, this suggests that despite their correlation, it was not inappropriate to use this collection of factors as control variables in the model. This highest variance inflation was found for $matchComm$, which is highly correlated with $coordReq$, $extraComm$, and $projectAge$, as shown in Table 4. This correlation was largely due to the fact that $coordReq$ is the maximum value of $matchComm$, and therefore high correlation was likely as a result of variable construction.

Insert Table 4 about here

DISCUSSION

Software engineering is a difficult and risky proposition. Failures in communication often lead to failures in development which leads to overall project failure. Therefore, if we can develop an automated tool or metric that helps direct communication we can reduce the risk of the software engineering project. The original work of STC laid the foundation for understanding the linkage between communication, dependencies, and performance, but the aggregate nature of the metric limited its usefulness.

This work took the original model of STC and expanded it from the team level to individual level (model 2) where it was found that an individual who matched their communication with coordination requirements was able to resolve defects assigned to them faster. This new metric is much more useful for individual developers and product manager because it is based primarily on the actions of an individual. We note that the comparative coefficient within the regression model of ISTC was of a greater magnitude than that of STC, showing that the communication and technical dependencies of the individual have a much greater role in reducing time to resolve bugs than they do when examining the original model of STC at the team level.

Another side effect of evaluating the impact of STC and ISTC on the team on a post-hoc basis is that it is not possible to discern if the knowledge of STC and ISTC (and their related coordination requirements networks) by individual developers can impact the communication patterns of the participants in the community. For example, is it possible to use ISTC as a sort of a recommendation engine for developers to provide context about other individuals with whom they should communicate. While work has been done to create an end user tool to evaluate STC and visualize the networks of communication, it has only been used in a post-hoc manner

(Sarma, Maccherone, Wagstrom, & Herbsleb, 2009). Fortunately, the increasing accessibility and availability of integrated systems such as IBM's Rational Jazz environment and Google's Code project hosting and management system provide the potential for automated real time collection of the elements needed to calculate STC and ISTC. Once a tool is designed, the next step would be to evaluate how the use of such a tool changes the communication and participation patterns of the team.

Perhaps most importantly, this work has been able to differentiate between the impact of general communication and communication that is targeted to address coordination dependencies. It was found that not only did communication that matched coordination dependencies have a greater impact on reducing the time to resolve bugs in the selection of sub-projects, but also when the standardized betas were calculated for each of the independent variables that more of the variance in the project was explained by the matched communication than by the communication which is not matched with a coordination dependency. We take this result to be strong support for the STC hypothesis, and find no support for the centrality hypothesis.

This finding has multiple implications for developers and users of group awareness and management tools. Firstly, it indicates that it is not enough to simply create tools that increase the amount of communication in a team as there no guarantee that the communication will be directed along the lines of coordination dependencies. While communication that was not aligned with coordination requirements generally did not increase the time to resolve the bugs, it did little to improve performance either.

Secondly, it indicates that when evaluating a team or a virtual organization, it is more important that communication is between the proper parties than that communication is happening. For example, a community member may be very active on discussion forums and mailing lists, but may have problems getting actual work done because those conversations do not address the actual technical coordination issues at hand. For a project manager this may mean scheduling proper face-to-face team between developers rather than holding large group meetings. The use of metrics such as STC and ISTC could quickly identify the individuals with the most knowledge of the community and who were commenting about the technical issues without the need to take the time to read and understand every message of the discussion. Thus, there exists the possibility that these metrics could easily be used as tools for expertise finders in online communities.

There is also the potential of STC and ISTC to be used in management tools, possibly as part of a dashboard like system. The finding that communication that aligns with coordination dependencies has a greater beneficial effect at task performance than other communication provides a relatively easy metric for managers to better understand the efficiency of their employees. In such a system a manager could quickly get a view of the dependencies of the system and see if the individuals are communicating in ways that are aligned with the dependencies. Furthermore, if there is frequent communication that has no corresponding coordination requirement it may be a sign of expertise that resides outside of the team that needs to be brought into the team from and outside team or organization.

Finally, this work has replicated the original findings that showed that when a software team communicates along the lines of technical dependencies the performance of the team increases (Cataldo et al., 2006). In contrast to the original study which relied on highly manicured data from a controlled, commercial software engineering environment, we found the same results using an Open Source data set that was automatically collected (although some manual intervention was required to parse out the projects and reconcile user accounts across different community systems). Most interestingly, many of the developers within GNOME are volunteer developers working on the project during their free time. These results indicate that both professional and volunteer based organizations receive similar benefits from communicating along the lines of technical dependencies and suggest it may be possible to use STC as a tool to help understand organizational performance in a wide variety of different contexts involving volunteer user generated content, for example, volunteer systems such as Wikipedia. This is especially interesting given the previous findings of Kittur and Kraut that found that adding additional editors to an article on Wikipedia only improved quality when appropriate coordination methods were used (Kittur & Kraut, 2008).

We note that the regression model we used to validate these results, while fitting the data quite well, tends to have difficulty with bugs at the extremes of the range of time that bugs were open. For example, many bugs that were closed nearly immediately were predicated to take much longer, while the model tends to under-predict resolution time for extremely long-lived bugs. While our goal was to perform the analysis using wholly automated data collection and analysis methods, an in-depth human review of these extremes may prove beneficial in understanding the errors in the model at these levels. From a software engineering point of view, the bugs that are closed quickly – within a day – are not our primary interest; however, the long lived bugs do pose an issue and challenge.

CONCLUSION

Software engineering remains a difficult endeavor with a myriad of hidden technical dependencies that can be addressed by communication between team members. This work shows that communication is about more than just sending emails to team members and ensuring that everyone knows what pieces of software you developed. For maximum benefit developers should be aware of technical dependencies within the project and communicate to address those dependencies, something that, unfortunately, most developers fail to do (Cataldo et al., 2006). This work has shown that individuals who merely communicate a lot without regard to technical dependencies receive no benefits from their communication, but that understanding with whom they should be in communication with to address coordination dependencies is more important for the performance of the organization and individual.

For teams designing and building complicated projects, such as the engineering and software domains, this work has shown that not only can STC be used as a framework to direct communication in the organization, but also that it can be individualized to better direct individual communication within the organization. Furthermore, the relationship between communication and coordination requirements reinforces the notion that such a tool could yield great benefits to the organization.

REFERENCES

Bayerl, P. S., & Lauche, K. 2008. Coordinating high-interdependency tasks in asymmetric distributed teams. In ***Proceedings of the ACM 2008 conference on Computer supported cooperative work***: 417-426, San Diego, CA, USA: ACM.

Becker, M. C. 2001. Managing Dispersed Knowledge: Organizational Problems, Managerial Strategies, and Their Effectiveness. ***Journal of Management Studies***, *38*(7): 1037-1051.

Bird, C., Gourley, A., Devanbu, P., Gertz, M., & Swaminathan, A. 2006. Mining email social networks. In ***Proceedings of the 2006 International Workshop on Mining Software Repositories***: 137-143, Shanghai, China: ACM.

Brooks, F. P. 1995. ***The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*** (2nd ed.), Boston, MA, USA: Addison-Wesley Professional.

Carley, K. M., & Krackhardt, D. 1998. A PCANS model of structure in organization. In ***1998 International Symposium on Command and Control Research***, Presented at the International Symposium on Command and Control Research.

Cataldo, M., & Herbsleb, J. D. 2008. Communication networks in geographically distributed software development. In ***Proceedings of the ACM 2008 conference on Computer supported cooperative work***: 579-588, San Diego, CA, USA: ACM.

Cataldo, M., Herbsleb, J. D., & Carley, K. M. 2008. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In ***Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement***: 2-11, Kaiserslautern, Germany: ACM.

Cataldo, M., Wagstrom, P. A., Herbsleb, J. D., & Carley, K. M. 2006. 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***: 353-362, Presented at the 2006 Conference on Computer Supported

Cooperative Work, Banff, Alberta, Canada: ACM Press.

Conway, M. 1968. How Do Communities Invent? ***Datamation***, *14*(5): 28-31.

Cummings, J. N., & Kiesler, S. 2008. Who collaborates successfully?: prior experience reduces collaboration barriers in distributed interdisciplinary research. In ***Proceedings of the ACM 2008 conference on Computer supported cooperative work***: 437-446, San Diego, CA, USA: ACM.

Curtis, B., Soloway, E., Brooks, R., Black, J., Ehrlich, K., & Ramsey, H. 1986. Software psychology: The need for an interdisciplinary program. ***Proceedings of the IEEE***, *74*(8): 1092-1106.

Druskat, V. U., & Kayes, D. C. 2000. Learning versus Performance in Short-Term Project Teams. ***Small Group Research***, *31*(3): 328-353.

Ducheneaut, N. 2005. Socialization in an Open Source Software Community: A Socio-Technical Analysis. ***Computer Supported Cooperative Work***, *14*(4): 323-368.

Fogel, K. 2005. ***Producing Open Source Software***, Sebastapol, CA: O'Reilly & Associates.

Gall, H., Hajek, K., & Jazayeri, M. 1998. Detection of Logical Coupling Based on Product Release History. In ***14th IEEE International Conference on Software Maintenance***, Presented at the International Conference on Software Maintenance, Bethesda, MD: IEEE Press.

German, D. 2004. The GNOME project: a case study of open source, global software development. ***Software Process: Improvement and Practice***, *8*(4): 201-215.

Grinter, R. E., Herbsleb, J. D., & Perry, D. E. 1999. The geography of coordination: dealing with distance in R&D work. In ***Proceedings of the 1999 ACM SIGGROUP International Conference on Supporting Group Work***: 306-315, Presented at Group 1999, Phoenix, AZ, USA: ACM.

Halloran, T., & Scherlis, W. 2002. High Quality and Open Source Practices. In ***2nd Workshop on Open Source Software Engineering***, Presented at the 2nd Workshop on Open Source Software Engineering, Orlando, Florida.

Herbsleb, J. D., Mockus, A., Finholt, T. A., & Grinter, R. E. 2001. An empirical study of global software development: distance and speed. In ***Proceedings of the 23rd International Conference on Software Engineering***: 81-90, , Toronto, Ontario, Canada: IEEE Computer Society.

Hertel, G., Niedner, S., & Hermann, S. 2003. Motivation of Software Developers in Open Source Projects: An Internet-based Survey of Contributors to the Linux Kernel. ***Research Policy***, *32*(7): 1159-1177.

Hinds, P., & Mcgrath, C. 2006. Structures that Work: Social Structure, Work Structure and Coordination Ease in Geographically Distributed Teams. In ***Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work***: 343-352, Presented at the 2006 Conference on Computer Supported Cooperative Work, Banff, Alberta, Canada: ACM Press.

Hinds, P. J., Carley, K. M., Krackhardt, D., & Wholey, D. 2000. Choosing Work Group Members: Balancing Similarity, Competence, and Familiarity. ***Organizational Behavior and Human Decision Processes***, *81*(2): 226-251.

de Icaza, M. 2002, April 9. The Story of the GNOME Project, http://primates.ximian.com/~miguel/gnome-history.html, January 10, 2010.

Jackson, P. J. 1999. Organizational change and virtual teams: strategic and operational integration. ***Information Systems Journal***, *9*(4): 313-332.

Jarvenpaa, S. L., & Leidner, D. E. 1999. Communication and Trust in Global Virtual Teams. ***Organization Science***, *10*(6): 791-815.

Kittur, A., & Kraut, R. E. 2008. Harnessing the wisdom of crowds in wikipedia: quality through coordination. In ***Proceedings of the ACM 2008 conference on Computer supported cooperative work***: 37-46, San Diego, CA, USA: ACM.

Koch, S., & Schneider, G. 2002. Effort, co-operation and co-ordination in an open source software project: GNOME. ***Information Systems Journal***, *12*(1): 27-42.

Lipnack, J., & Stamps, J. 2000. ***Virtual Teams: People Working Across Boundaries with Technology*** (2nd ed.), New York: Wiley.

Lopez-Fernandez, L., Robles, G., & Gonzalez-Barahona, J. M. 2004. Applying Social Network Analysis to the Information in CVS Repositories. In ***Proceedings of the 2004 International Workshop on Mining Software Repositories***, Presented at the 2004 International Workshop on Mining Software Repositories, Edinburgh, Scotland, UK: IEEE.

Merrill, J., Caldwell, M., Rockoff, M., Gebbie, K., Carley, K., & Bakken, S. 2008. Findings from an Organizational Network Analysis to Support Local Public Health Management. ***Journal of Urban Health***, *85*(4): 572-584.

Mockus, A., Fielding, R., & Herbsleb, J. 2002. Two case studies of open source software development: Apache and Mozilla. ***ACM Transactions on Software Engineering and Methodology***, *11*(3): 309-346.

Montoya-Weiss, M. M., Massey, A. P., & Song, M. 2001. Getting It Together: Temporal Coordination and Conflict Management in Global Virtual Teams. ***The Academy of Management Journal***, *44*(6): 1251-1262.

Ren, Y., Carley, K. M., & Argote, L. 2006. The Contingent Effects of Transactive Memory: When Is It More Beneficial to Know What Others Know? ***Management Science***, *52*(5): 671-682.

Sarma, A., Maccherone, L., Wagstrom, P., & Herbsleb, J. 2009. Tesseract: Interactive Visual Exploration of Socio-Technical Relationships in Software Development. In ***Proceedings of the 2009 International Conference on Software Engineering***, Presented at the International Conference on Software Engineering, Vancouver, BC.

Sosa, M. E., Eppinger, S. D., & Rowles, C. M. 2004. The Misalignment of Product Architecture and Organizational Structure in Complex Product Development. ***Management Science***, *50*(12): 1674-1689.

The GNOME Foundation. 2010, January. GNOME: The Free Software Desktop Project, http://www.gnome.org/, January 6, 2010.

Townsend, A., Demarie, S., & Hendrickson, A. 1998. Virtual teams: Technology and the workplace of the future. ***Academy of Management Executive***, *12*(3): 17-29.

Valetto, G., Helander, M., Ehrlich, K., Chulani, S., Wegman, M., & Williams, C. 2007. Using Software Repositories to Investigate Socio-Technical Congruence in Development

Projects. In ***2007 Workshop on Mining Software Repositories***, Presented at the Mining Software Repositories 2007, Minneapolis, MN, USA.

Yamauchi, Y., Yokozawa, M., Shinohara, T., & Ishida, T. 2000. Collaboration with Lean Media: how open-source software succeeds. In ***Proceedings of the 2000 ACM conference on Computer supported cooperative work***: 329-338, Presented at the CSCW 2000, Philadelphia, Pennsylvania, United States: ACM.

Table 1. Correlations of Model Control Variables. Values in bold indicate correlations above 0.3.

	numDevs	***numDeltas***	***numComments***	***severity***	***projectAge***	***newBugs***	***devExperience***
numDeltas	**0.4456**						
numComments	**0.3281**	**0.5241**					
severity	0.0246	0.0365	0.0696				
projectAge	0.0055	0.1514	-0.0238	0.0287			
newBugs	0.2495	**0.3543**	0.2203	-0.0840	0.0619		
devExperience	0.0326	-0.0143	0.0239	0.0163	0.2274	-0.0032	
cvsCommits	0.0106	-0.0291	0.0420	-0.0145	-0.1736	0.0087	0.0103

Table 2. Regression models predicting time to resolve bugs using automatically collected data from a selection of projects in the GNOME community

Variable	Model 1			Model 2			Model 3		
	Estimate	Std Error	P-Value	Estimate	Std Error	P-Value	Estimate	Std Error	P-Value
Intercept	-0.4354	0.1266	0.0006	-0.3839	0.1261	0.0023	-0.4238	0.1281	0.0009
$numDevs$	0.1812	0.0141	<.0001	0.1763	0.0141	<.0001	0.1587	0.0143	<.0001
$numDeltas$	-0.0414	0.0027	<.0001	-0.0410	0.0027	<.0001	-0.0438	0.0027	<.0001
$numComments$	-0.0160	0.0019	<.0001	-0.0157	0.0019	<.0001	-0.0142	0.0019	<.0001
$severity$	-0.0581	0.0054	<.0001	-0.0606	0.0054	<.0001	-0.0642	0.0054	<.0001
$projectAge$	-0.0500	0.0087	<.0001	-0.0472	0.0086	<.0001	-0.0915	0.0098	<.0001
$newBugs$	0.6773	0.0023	<.0001	0.6759	0.0023	<.0001	0.6767	0.0023	<.0001
$devExperience$	-0.2322	0.0111	<.0001	-0.2297	0.0110	<.0001	-0.2119	0.0111	<.0001
$cvsCommits$	-0.2933	0.0453	<.0001	-0.3338	0.0459	<.0001	-0.1986	0.0466	<.0001
STC	-0.1646	0.0622	0.0081						
$ISTC$				-0.2547	0.0390	<.0001			
$coordReq$							0.0224	0.0017	<.0001
$matchComm$							-0.0227	0.0028	<.0001
$extraComm$							0.0031	0.0017	0.0706
	$R^2 = 0.7942, p < 0.0001$			$R^2 = 0.7945, p < 0.0001$			$R^2 = 0.7955, p < 0.0001$		

Table 3. Standardized Betas and Variable Inflation for Predictors in Model 3

Variable	Standard β	VIF
$numDevs$	0.0355	1.3368
$numDeltas$	0.0601	1.7473
$numComments$	0.0255	1.4470
$severityNum$	0.0338	1.0403
$projectAge$	0.0318	1.5006
$newBugs$	0.8994	1.1800
$devExperience$	0.0555	1.0963
$cvsCommits$	0.0125	1.1063
$coordReq$	0.0612	2.7498
$matchComm$	0.0425	3.4962
$extraComm$	0.0069	1.8929

Table 4. Correlation between coordination/communication variables in model 3 and control factors. Correlations not indicated are below 0.2.

	$projectAge$	$coordReq$	$matchComm$
$coordReq$	0.4961		
$matchComm$	0.4596	0.7465	
$extraComm$	0.3435	0.3945	0.6500

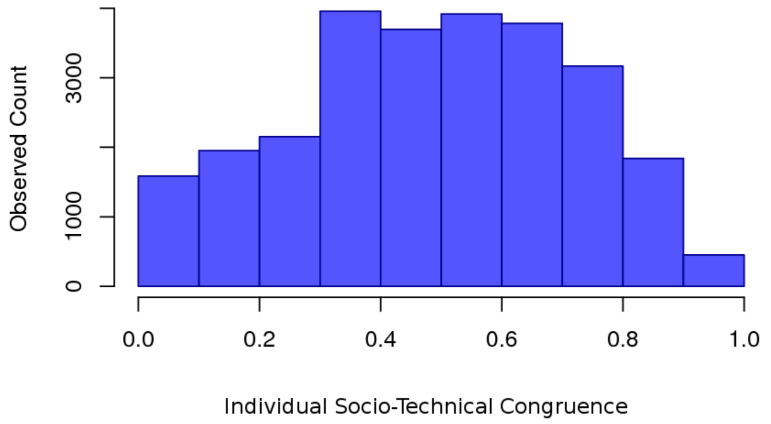

Figure 1. Distribution of ISTC across individuals and bugs within a subset of projects in the GNOME desktop environment.

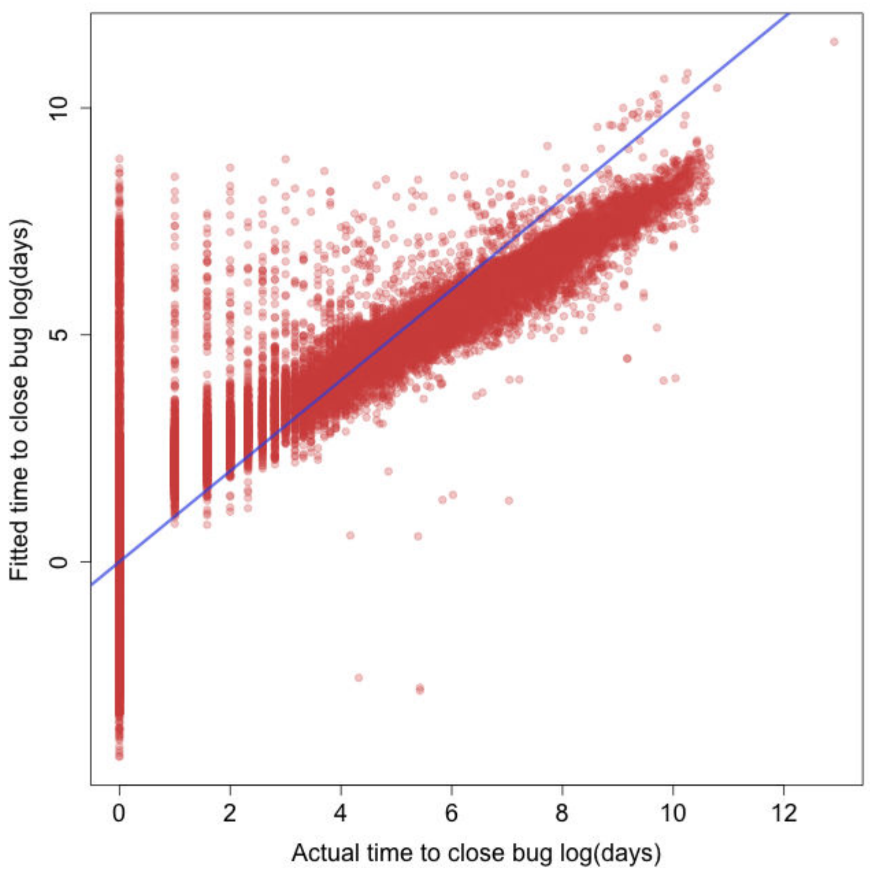

Figure 2. Fitted vs. actual time to close bugs from model 3