Congruence Might Have No Effect: A Study of Coordination in a Software Project

Journal:	*Transactions on Software Engineering*
Manuscript ID:	TSESI-2009-10-0257
Manuscript Type:	Special Issue on Socio-Technical Environments
Keywords:	D.2.14 Human Factors in Software Design < D.2 Software Engineering < D Software/Software Engineering, K.4.3.b Computer-supported collaborative work < K.4.3 Organizational Impacts < K.4 Computers and Society < K Computing Milieux, D.2.8 Metrics/Measurement < D.2 Software Engineering < D Software/Software Engineering, D.2.9.e Organizational management and coordination < D.2.9 Management < D.2 Software Engineering < D Software/Software Engineering, D.2.9.i Programming teams < D.2.9 Management < D.2 Software Engineering < D Software/Software Engineering

Congruence Might Have No Effect: A Study of Coordination in a Software Project

Irwin Kwan, *Member, IEEE,* Adrian Schröter, *Member, IEEE,* Daniela Damian,*Member, IEEE*

Abstract—Socio-technical congruence is an approach that measures coordination by examining the alignment between the technical dependencies and the social coordination in the project. We studied coordination in the IBM Jazz project, which consists of 147 developers over seven geographically distributed sites, and expected that high congruence would lead to successful software builds. We conducted an exploratory case study and explored this relationship by applying two congruence measurements: the existing congruence measure in the literature, and a new measure that overcomes limitations with existing measures. We discover that there is no relationship between socio-technical congruence and build quality, that overall congruence in the project is low, that even a large proportion of low congruence builds are successful, and that socio-technical gaps in successful builds are larger than gaps in failed builds. More in-depth analysis of the social and technical aspects in the Jazz project allows us to discuss in detail why congruence has no effect on build quality. Our findings call into question current conceptualizations of socio-technical congruence and have implications with respect to further improvements of measures of socio-technical congruence in software development.

Index Terms—Empirical software engineering, socio-technical congruence

1 Introduction

Coordinating the efforts of individuals working together on a team is necessary to build software systems. The complexity of current systems require contributions from tens or hundreds of people who may span multiple offices, cities, or even continents. To build such systems, we need to be able to ensure that the team is not only capable of developing components of a system, but also has the governance to be able to integrate the interdependent parts into a whole.

In this paper we describe a case study of socio-technical coordination in IBM's Jazz project. Our approach to investigating coordination is to examine the alignment between the technical dimension of work and the social relationships between team members. This alignment is called **socio-technical congruence** [1]. High socio-technical congruence has been shown to be a predictor of coordination success [1], [2], [3]. The mismatches between the social and technical dimensions, or **gaps**, also have been observed to increase resolution times for software activities, suggesting poor coordination in a software project. However, the quantitative effect of socio-technical congruence on software quality is still unclear.Additional quantitative studies are required to clarify this relationship.

We conducted an exploratory case study examining the effects of socio-technical congruence on the quality of a large software project at IBM. We sought to discover the relationship that congruence, and in turn, collaboration, has on the results of regularly-scheduled builds. A build result, which can be *error* or *OK*, indicates the relative health of the project up to that build. To measure socio-technical congruence we apply two different measures: a previously-published congruence approach [1], and a new weighted congruence approach that provides details about the size of a gap between a pair of individuals [4]. We also examine the Jazz project's processes and tools to identify any explanations of the relationship between congruence and health of a build that we find. Our results were somehow surprising and not what we expected from knowledge we had from the literature. Our findings on the socio-technical congruence and integration quality in Jazz are complemented by insights we obtained from other factors in the Jazz project that influence both the socio-technical congruence and integration quality. By using the weighted congruence measure we could investigate gaps in coordination at a finer-grain level of detail and unveil interesting patterns in the teams' coordination behaviour. Our case study adds to the scarce but needed empirical evidence about socio-technical congruence measures in the investigation of coordination in software projects. Finally, the insights we obtained by applying socio-technical congruence measurements in the Jazz context allows us to discuss useful further improvements to congruence measures.

The paper begins with background on the need for coordination in software projects and socio-technical congruence, and motivates the applicability of socio-technical congruence to our case study setting in Section 2. We also present background information on the project we investigated. We discuss related work in Section 3 and show the research objective in Section 4. We then describe in Section 5 the two measurements for socio-technical congruence that we used in our case study:

• *I. Kwan, A. Schröter, S. Marczak, and D. Damian are with the Department of Computer Science, University of Victoria, Victoria, British Columbia, Canada.*
E-mails: {irwink,schadr,smarczak,danielad}@cs.uvic.ca

an unweighted congruence measurement found in the literature [1], and a weighted congruence measurement from our own work [4]. We describe our research questions and methodology in Section 6, and outline the results in Section 7. Our discussion of these results in Section 8 include not only an answer to whether socio-technical congruence does matter to affect software quality, but also provides some explanations of the effect we observed, and which we based on more in-depth examination of socio-technical gaps and other factors in the Jazz environment. We conclude the paper by outlining the threats to validity in our study (Section 9) and possible avenues for future work (Section 10).

2 Background and Motivation

For the background of this paper, we discuss the reasons we need coordination in software projects, and how we can use socio-technical congruence to study how coordination affects software quality. We also present our case study of the IBM Jazz project, as a project that is interesting to study from both a coordination and a quality point of view.

2.1 The Need for Coordination

Software is extremely complex because of the sheer number of dependencies [5]. Large software projects have a large number of components that interoperate with one another. The difficulty in software product development occurs when making changes to these projects, because a change in one component of the software often requires changes in dependent components [6]. Because a single person's knowledge of a system is specialized as well as limited, that person often is unable to make the appropriate modifications in dependent components when a component is changed.

Coordination is defined as "integrating or linking together different parts of an organization to accomplish a collective set of tasks" [7]. In order to manage changes and maintain quality, developers must coordinate, and in software development, coordination is largely achieved by communicating with people who depend on the work that you do [8].

We study the relationship between team coordination and the success of code integration. In our study, code is integrated regularly into a software build, and thus a successful software build can be viewed as the outcome of good coordination. A failed build, on the other hand, demotivates software developers [9], [10] and destabilizes the product [11]. While a failed build is not necessarily a disaster, it slows down work significantly and is considered extremely undesirable. A build result thus serves as an indicator of the health of the software project up until that point in time. If the developers successfully coordinate the integration of code between the previous build and the upcoming build, then the build should succeed.

Thus, in order to effectively build software, a developer should coordinate closely with individuals whose technical dependencies affect his work. This brings forward the idea of aligning the technical structure and the social interactions [12], leading us to the foundation of socio-technical congruence.

2.2 Socio-technical Congruence

Socio-technical congruence is defined as the match between the coordination needs established by the technical domain and the actual coordination activities carried out by project members. Socio-technical congruence was first proposed by Cataldo, et al. [1]. A **coordination need** indicates that one person and another person should be coordinating, based on the technical dependencies on the project. A coordination need is determined by analysing the assignments of people to a technical entity, such as a source code module, and the technical dependencies among these technical entities. For example, such a technical dependency may be a set of modules where, in the case that one module changes, other modules must be updated accordingly to reflect the change. The concept, intuitively, is that if one person works on one module of the project, and his colleague works on another component, but the two components are interdependent, then that person should be coordinating with his colleague.

If two individuals have a coordination need, but do not coordinate, then there is a **gap** between these two individuals. A gap suggests the existence of a coordination problem. One of the goals of socio-technical congruence is to minimize the number of gaps, either by maintaining good coordination between individuals who have a coordination need, or by reducing the number of technical dependencies in the project and therefore reducing the coordination needs [13].

What socio-technical congruence offers is a fine-grained approach to measure the coordination quality in a project [1]. We can use this measurement to identify the effect of socio-technical congruence on software build quality.

2.3 The Jazz Project

In order to study socio-technical congruence, we need a project that has high coordination needs, as well as a large amount of recorded communication. The Jazz project by IBM fulfils these requirements.

The IBM Jazz project is a large software development project working on an enterprise-level software development environment. It incorporates the integrated development environment, the source-code control system, the issue-tracking system, and collaborative development tools.

The following qualities make Jazz an attractive case in which to study socio-technical coordination: (1) the distributed team, (2) the iterative process, and (3) the community involvement. Distribution makes more difficult to coordinate work effectively [14], [15] which

the process is supposed to mitigate by enhancing communication among project members [16]. This therefore forces developers to record, explicitly, communication across sites. Jazz's iterative process encourages developers to "build early and often". In addition to supporting continuous, daily builds, Jazz does frequent integration builds, providing a large amount of data to analyse. The encouraged community involvement and the openness to the community poses extra stress on the developers because of the increased information input they get [17].

The Jazz team is a large distributed team and uses the Jazz platform for development. The Jazz development involves distributed collaboration over 16 different sites located in the United States, Canada, and Europe. Seven sites are active in Jazz development and testing. There are 151 active contributors working in 47 teams at these locations, where contributors belong to multiple teams. Each team is responsible for developing a subsystem or component of Jazz. The team size ranges from 1 to 20 and has an average of 5.7 members. The number of developers per geographical site ranges from 7 to 24 and is 14.8 in average.

The project uses the *Eclipse Way* development process [16]. It defines six-week iteration cycles, which are separated into planning, development and stabilization activities. A project management committee formulates the goals and features for each release at the beginning of the each iteration, and *work item*s represent assignable and traceable tasks for each team. Jazz's process encourages frequent building, including continuous, nightly, weekly, and integration builds.

One unique aspect of Jazz is its "open-commercial" development model. Jazz is a commercial product developed by IBM, but has a publicly-accessible issue-tracking and reports system. This open-commercial model causes a large amount of information to flow into the team from the community. Communication between individuals is done face-to-face, through the issue-tracking comments, through instant messaging, through Internet Relay Chat, and by phone. Personal e-mail is discouraged. The team uses a mailing list to deliver announcements, such as server maintenance schedules.

3 Related Work

Socio-technical congruence was proposed initially as a fine-grained measure of coordination that can be used to diagnose coordination problems in a software development team [1]. The original conceptualization of socio-technical congruence is in Conway's Law, which observed that product architecture reflects organizational structure [18]. Socio-technical congruence has been recognized as an important element of product design in the management sciences field [19]. The socio-technical congruence community goes one step further and declares that not only does this alignment happen as a consequence of product architecture, but that it is in fact desired [13], [20]. Unfortunately there are only a handful of empirical research results that discuss the effects of the socio-technical congruence approach on software teams.

First, we review research on studies of coordination in software engineering, and then we related work on socio-technical congruence.

3.1 Coordination in Software Teams

Research in software-engineering coordination has examined interactions among software developers [21], [22], how they acquire knowledge [23], [24], and how they cope with issues including geographical separation [15], [25], [26].

The ability to coordinate has been shown as an influential factor in customer satisfaction by Kraut and Streeter [8]. These results are corroborated by Faraj and Sproull [27], who suggest that coordinating experts effectively improve the capability to produce high quality work.

It has been shown that software developers spend much of their time communicating [28], [29], emphasizing communication as an important source of information in coordination. Because developers face problems when integrating different components from heterogeneous environments [30], developers engage in direct or indirect communication, either to coordinate their activities, or to acquire knowledge of a particular aspect of the software [24]. Herbsleb, et al. examined the influence of coordination on integrating software modules through interviews [31], and found that processes, as well as the willingness to communicate directly helped teams integrate software. Redmiles, et al. [32] studied a software team engaging in implicit communication and found that these implicit communications are important to avoid collaboration breakdowns and delays; they also found that architecture affected the awareness network. Ko, et al. investigated software teams [33] and found that developers were identified as the main source of information about code issues. Wolf et al. [34] described a study in which they used properties of social networks to predict the outcome of integrating the software parts within teams [35]. This prior work establishes the fact that developers communicate heavily about technical matters.

It is well-known that coordination of software teams becomes more difficult as the distance between people increases. Herbsleb, et al. [36] has indicated that work that is done over distance can take up to 2.5 times longer than colocated work, Several studies at Microsoft [37], [38] showed that different kinds of distance between people that work together on a binary determine the binaries failure proneness. It has also been shown that differences in time zones affect the number of defects in software projects [39].

Although distance has been identified as a challenge, advances in collaborative development environments are enabling people to overcome challenges of distance. One recent study of the IBM Jazz project that uses Jazz for

collaboration has shown that the task completion time is not as long as previously thought [40]. Examples of technology that has helped enable distributed collaboration include topic recommendations and notification [21] and instant messaging [41]. Processes are also adapting to the fast pace of software development: the Eclipse way [16] is one such process that emphasizes placing milestones at fixed intervals and community involvement to improve the product. However, the frequent communication that technology enables has also caused problems [10], [32].

Researchers have conceptualized coordination in many different ways, including direct communication, indirect communication, proximity, and awareness [1], [23], [25], [42]. Socio-technical congruence allows us to determine if an organization has good coordination practices.

3.2 Effects of socio-technical congruence

Current research suggests that attaining a high level of socio-technical congruence is very beneficial to an organization. Evidence shows that higher congruence leads to faster completion of modification requests in a software development project [1], [3]. Since distribution of project always created additional challenges, previous looked into the effect of gaps on distributed development. Ehrlich et al. [2] found that the presence of gaps increased the number of code changes.

Socio-technical gaps have been found to be an issue not only because they lower the congruence and thus lowering productivity [1], but they become especially problematic in the context of distributed development [43]. Thus, researchers like Valetto et al. [20] have proposes remedial actions when socio-technical congruence gaps are discovered. Examples of actions include closing a gap by augmenting coordination, eliminating the gap by refactoring software, and eliminating the gap by rearranging schedules.

The general strength of measuring socio-technical congruence depends on the way the social and the technical parts are conceptualized. Although, communication is believed to help people coordinate [44] it is not the only way to describe the social dimension. For instance, Cataldo et al. [1] evaluated congruence in the context of a software development using different representations of actual coordination, including geographical proximity, IRC communication, and issue-tracking comments; these factors correlate with the resolution time of modification requests. There are also variations in the way technical dependencies can be handled. Cataldo et al. [3] used differing ways to measure architectural dependencies and found that the congruence values computed using a "files changed together" dependencies are more reliable than call graph dependencies [45], [46].

Thus, we see (1) that there is a need to coordinate effectively in software development across geographical boundaries, (2) that socio-technical congruence is a useful mechanism for studying coordination, and (3) that there is a need to coordinate across technical dependencies.

4 Research Objective

Our research objective in this paper is to study coordination in a large, distributed software team using a socio-technical approach. Specifically, we investigate the relationship between socio-technical congruence and build quality in the Jazz project. We study the relationship between congruence and build quality using two different congruence measurements, one of which we designed to address some limitations of the existing measure. We also investigate the relationship between socio-technical congruence, technical factors, and social factors that may occur in the Jazz project.

First, we present two approaches to calculate congruence: an unweighted congruence measurement developed by Cataldo and et al. **??**, and a weighted congruence measurement from our own work [4]. We then present our research questions and methodology, followed by our study results.

5 Calculating Congruence

In this section, we present two techniques to calculate socio-technical congruence. The first measure was originally proposed by Cataldo et al. [1], but bears some limitations. The second measure [4] addresses one of these limitations.

5.1 Technical Entities and Relationships

Socio-technical congruence has focused on the source code file as the technical artifact that has dependencies to another artifact [1], [2], [3], although some work has also examined socio-technical congruence using a requirement as the technical item of interest [22], [42], [47] and other proposals have used tasks as the technical item of interest [34]. A **technical entity** is an entity in a project that can be worked on by a person. Examples of a technical entity include a source code file, a compiled binary, a requirement, a task, or a bug. The choice of what a technical entity entails depends on what it is that a study intends to investigate. In a project, a person is assigned to one or more technical entities.

At the core of socio-technical congruence is the concept of a technical dependency. A **technical dependency** is a type of dependency between two technical entities. Examples of possible technical dependencies are listed in Box 1.

A **coordination need** indicates that one person and another person should be coordinating, based on the assignment of each person to a technical entity, and the technical dependencies between the entities.

Actual coordination indicates how people in the organization are actually coordinating. Note that, despite

Box 1 Examples of technical dependencies

- Requirements that depend on each other [22], [47]
- Source code modules that are changed together in a change set [1], [3]
- Source code that has a call-graph dependency [45]
- Tasks that depend on other tasks [34]

the nomenclature "coordination" used in previous work [1], the actual coordination matrix does not need to be coordination at all, but merely a relationship of interest between two people in an organization. Generally, one would want to choose relationships that are of interest to the performance of the organization, hence favouring the selection of relationships such as "communication". Examples of relationships appear in Box 2. Actual coordination may also be represented as a composite of relationships, such as in Ehrlich, et al [2].

5.2 Calculating Socio-Technical Congruence

This section repeats the formal definition of socio-technical congruence as published by Cataldo, et al. [1]. Given m people, and n technical entities to examine, the coordination needs are calculated as

$$CN = A \times D \times (A)^t$$

where A is an $m \times n$ assignment matrix that indicates that a person is assigned to a technical entity of the project, and D is a $n \times n$ technical dependency matrix that indicates that a technical entity in the project is dependent on another entity. The result is CN, an $m \times m$ coordination needs matrix that indicates who in the project should be coordinating with whom.

Once the coordination needs matrix is calculated, congruence is calculated by comparing this matrix to an actual coordination matrix. If an edge exists in the actual coordination matrix, and the same edge exists in the coordination needs matrix, then there is full congruence on this edge. The socio-technical congruence index is calculated as

$$congruence = \frac{\sum_{i=1}^{m}\sum_{j=1}^{m} CN_{ij} AC_{ij}}{\sum_{i=1}^{m}\sum_{j=1}^{m} CN_{ij}}$$

Box 2 Examples of actual coordination

- Communication, where Person A communicates with Person B [1], [2], [3], [48].
- Location, where Person A is in the same location as Person B [1], [2].
- Team structure, where Person A is in the same team as Person B [1].

where AC is an $m \times m$ matrix representing actual coordination. If $i = j$, then ignore the value; congruence ignores diagonals. If may be the case that the actual coordination matrix may involve more people than appear in the assignment matrix; these additional people are dropped.

The resulting congruence value is a number between 0 and 1, where 0 indicates no congruence, and 1 indicates full congruence.

5.3 Limitations of the Current Socio-technical Congruence Calculation

Unfortunately, the measure of congruence presented by Cataldo, et al [1] bears some limitations. One limitation is that indirect coordination is not considered Another limitation is that coordination between individuals is either present, or not present.

Indirect coordination is not considered. Although limited, studies show that high congruence correlates with higher performance. However, Marczak et al. [22], [47] identified projects that delivered requirements on time despite the existence of gaps. Findings made by Ehrlich, et al. [2] suggests that brokers may be able to mitigate the effect of gaps. There may be communication structures in the organization that compensates for a lack of congruence [14].

Literature in other areas, namely social network analysis and management science supports the idea that gaps are not necessarily a liability, but simply a reality [45], [49], [50].

Coordination between individuals is either present, or not present. Current conceptualizations of coordination in socio-technical congruence indicate that a relationship is either present, or not present, with no values indicating relative strengths. As developers clearly have multiple dependencies [32], [51], one would presume that those pairs with more coordination needs would need to coordinate more. The problem is that current socio-technical congruence provides no way to observe which pairs have high coordination needs. One study uses a composite measure to represent actual communication but the final value is still dichotomized [43]. Valetto et al. [52] and Kwan et al. [4] suggest that the dependencies, social and technical, should be weighted. They suggest that the number of changes made to the same artifact, or the number of dependencies of a source file onto another are possible candidates for weights, but do not evaluate their proposed changes.

Thus, we believe that one of our objectives is not to eliminate every gap, but rather to identify which gaps in a matrix have high coordination needs and which gaps are not satisfied by actual coordination. In the next section, we propose a new method to calculate socio-technical congruence to address this limitation.

5.4 A Weighted Congruence Measurement

In this section we present an extension to congruence that addresses the limitation that coordination between

individuals is either present, or not present, from our own work [4]. The intention is that this improvement will provide more information about coordination behaviour than the current congruence measure.

5.4.1 Weighted Coordination Needs

Our measurement weighs each entry in the task assignment, task-dependency, and actual coordination matrices between 0 and 1. The Weighted Assignment matrix is a $m \times n$ matrix where m is the number of selected people and n denotes the number of selected entities. Each entry in the matrix defines the strength of the connection between a person i and an entity j.

The weighted dependency matrix is an $n \times n$ matrix where n denotes the number of selected entities. Each entry in the matrix describes the strength of the relation between two entities.

The weighted dependency matrix may show the ratio of dependencies a entity has with another over the number of total dependencies that entity has on others. For example, a entity strongly coupled with another has a high entity dependency.

To calculate the coordination needs matrix, we use the same formula proposed by Cataldo, et al. [1], using the weighted values in the task assignment A' and task dependency D' matrices.

$$CN' = A' \times D' \times (A')^t$$

This calculation will yield the weighted coordination needs matrix CN', which is an $m \times m$ people by people matrix (Figure 1(a)). The coordination needs matrix tells you how strong the relations between people are supposed to be.

5.4.2 Weighted Actual Communication

Weighted actual communication is a relative value that indicates the strength of a relationship between two individuals (Figure 1(b)). A social network such as a communication network can be weighed according to the amount of ongoing communication. For example, one such weighing scheme may work as follows:

1) For a communication network, identify the largest value of communication between two individuals.
2) Divide all other values by the largest one.
3) The result is a number between 0 and 1, where 1 indicates the largest value of communication in the network, and 0 is no communication.

Other relationships that may be used as actual coordination are the amount of distance between individuals and how frequently they meet with other members in their team.

5.5 Identifying Gaps in Socio-Technical Congruence

Currently, there is no formal technique to identify which pairs of individuals have a gap in congruence. We complement our weighted congruence measure with a technique to generate a *lack-of-coordination matrix*. This matrix not only identifies the gaps between a coordination needs matrix and an actual communication matrix, but can also identify the magnitude of these gaps (Figure 1(c)). This lack-of-coordination matrix can be used with both the unweighted congruence approach and the weighted congruence approach.

$$g_{ij}(CN'_{ij}, AC'_{ij}) = \begin{cases} 1, & CN'_{ij} - AC'_{ij} > 0 \\ 0, & \text{otherwise} \end{cases}$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, m$, where g_{ij} is the gap value. A value of 1 indicates that there is a gap, and a value of 0 indicates that there is no gap. The value of g_{ij} is the **gap size** between the pair ij.

The lack-of-coordination matrix illustrates situations where the proportion of communication exceeds the expected proportion of coordination needs. If a value is negative, then there was more coordination than requested through the CN' matrix, and this particular connection is *overloaded*. If the value is positive, then there is not enough coordination to satisfy the information needs requested by the CN' matrix. Although neither situation may necessarily result in a problem, we believe that there may be issues that result from overload or lack of coordination.

5.6 Weighted Congruence Index

To convert the lack-of-coordination matrix to a single socio-technical congruence measurement, we apply:

$$\text{congruence} = 1 - \frac{\sum_{i=1}^{m}\sum_{j=1}^{m} g_{ij}}{\sum_{i=1}^{m}\sum_{j=1}^{m} CN'_{ij}}$$

for each i and each j, unless $i = j$.

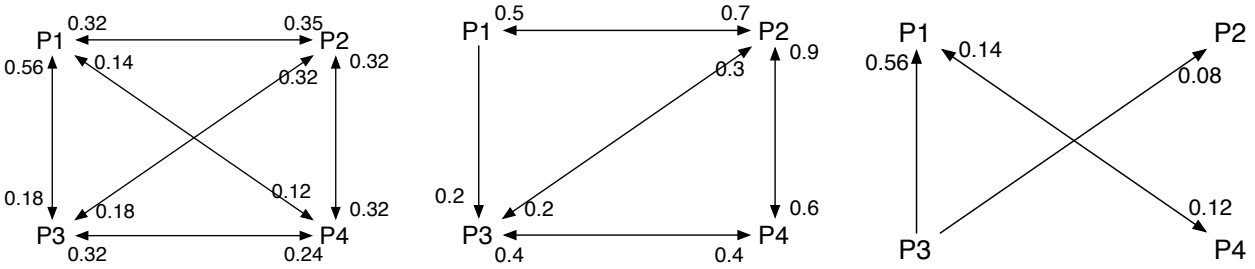
This value is an overall level of the congruence in the current network.

5.7 Benefits of Weighted Congruence

Using a weighted congruence model allows us to deal with situations that are not handled with unweighted congruence. We can investigate relationships between people and the technical entities in finer-grained detail, including situations where people have multiple coordination needs with others. Using the weighted congruence approach allows a person diagnosing the organization to benefit from *locality* and *relationship strength*.

Locality allows us to identify which area of a network has a gap. The lack-of-coordination matrix identifies which pairs of individuals do not fulfill coordination needs. Although the original model does give us some limited locality in the sense of identifying congruence gaps, it is far more coarse grained.

Relationship strength indicates how strong the relationship between two developers is, and mitigates



(a) Coordination needs derived from assignments and dependencies (b) The coordination that happened (c) The coordination that is lacking

Fig. 1. Comparing coordination needs with actual coordination to find lacking coordination.

the limitation that "coordination between individuals is either present, or not present". Relationship strength allows us to identify pairs of people who have multiple coordination needs, and therefore who need to coordinate more often with each other. Using relationship strength allows us to investigate coordination in more detail because we can investigate pairs with strong dependencies—these people also tend to be key individuals in a team.

By allowing better locality and relationship strength, we believe that this weighted model provides a more in-depth way to study congruence.

In the next section, we discuss how in our methodology we used these two measurements to investigate the relationship between socio-technical congruence and build quality.

6 Research Methodology

In this section, we present our research questions, our socio-technical conceptualizations in Jazz, and our analysis techniques.

6.1 Research Questions

We have established previously that coordination is essential in software development, and that high socio-technical congruence, which is one measurement of coordination quality, corresponds to high developer productivity. We also know that builds are relevant to the Jazz team; in an interview with one of the Jazz managers, he stated "Build quality is almost code quality". Specifically, the build outcome is either "error", which indicates that there was either a compilation error or a JUnit test suite error, "OK", which indicates that no errors were found, and "Warning", which indicates that there were warnings returned by the compiler or the JUnit test suite. Thus, an "OK" build implies good software quality. We therefore test the relationship between socio-technical congruence and build quality.

Research Question 1: Does socio-technical congruence have an effect on build quality in Jazz?

Our hypothesis is that a team that has high socio-technical congruence will experience fewer error builds.

In a large project such as Jazz, a developer has multiple dependencies with others. Thus, we expect that a pair of individuals who have high coordination needs will communicate more frequently than a pair of individuals with low coordination needs. By using the gap size we can investigate if developers coordinate more frequently when they have multiple coordination needs. Pairs who communicate often with those they have strong coordination needs will have a smaller gap size than pairs who do not communicate often.

Research Question 2: Does gap size have an effect on build quality in Jazz?

Our hypothesis is that the average gap size in an OK build will be smaller than the average gap size in an error build.

The Jazz project has many characteristics that may influence the amount of coordination in the project, such as the Jazz environment's collaboration-enhancing features, the geographical separation of its team members, and the communication media the team members use.

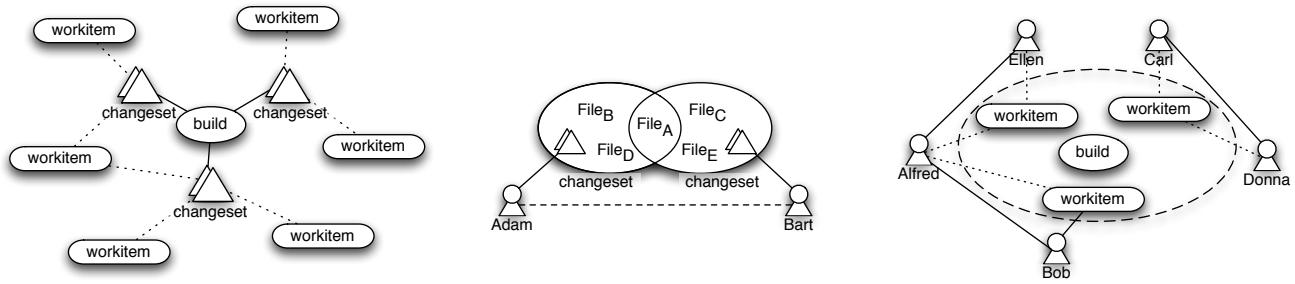
Research Question 3: What social and technical factors in Jazz affect congruence and build quality?

We pose this open question to explore possible explanations for the results of the other two research questions.

6.2 Data Description and Collection

In collaboration with IBM, we acquired a copy of the Jazz database containing information between June 2007 and June 2008. The data contains the following entities:

- *Build.* A build is a compilation of the software to form a working unit. Jazz performs team-based continuous builds, nightly builds, weekly builds, and integration builds that incorporate packages from each team. A build contains the work from one or more work items, and these work items are



(a) Relationship among Builds, Change Sets, Files, and Work Items

(b) Relationship among Builds, Change Sets, Files, and Work Items

(c) Constructing an Actual Coordination Instance

Fig. 2. Constructing Socio-technical Congruence Matrices in Jazz

not necessarily unique from build to build. The Jazz database does not keep a full record of every build over time. The Jazz database contains 533 builds, but we consider only builds that have technical dependencies. We also remove every build whose result is warning and consider only OK and Error builds. The resulting data subset contains 197 builds.

- *Work item.* A work item is a description of a unit of work to be done. A work item can be assigned a type of *task*, *enhancement*, or *defect*. The data subset contains 9344 work items.
- *Comment.* A comment is written text authored by a developer that is about a particular work item. Multiple comments may be attached to a single work item. Comments are not threaded—that is, we do not know if a comment specifically references an existing comment in the work item discussion. The data subset contains 54083 comments.
- *Change set.* A change set is a collection of code changes to a number of files. A change set is generated by a single author, and is related to one work item. A single work item may have multiple change sets. There are 9344 relevant change sets in Jazz.
- *Source code file.* A file in Jazz contains source code and are included in change sets. Over time, a file may be associated with multiple change sets.

We illustrate the relationship between builds, change sets, files, and work items in Figure 2(a).

In addition to quantitative data, we also conducted interviews with members of the Jazz team to learn about the context of the development team. We also investigated work items, comments, and reports available on the `jazz.net` web site.

6.3 Analysis Methods

To study whether congruence matters, we use the two congruence measurements presented in Section 5 for our analysis: Cataldo's unweighted congruence measure and the weighted congruence measure. To use congruence, we must conceptualize coordination needs between people as well as actual coordination.

6.3.1 Conceptualizing Coordination Needs

In the Jazz project, we conceptualize technical dependencies by examining files that were changed together for a build. This "files changed together" approach was used previously by Cataldo et al. [3] and was determined to be more robust than dependencies based on call graphs. We compute coordination needs matrix as follows (Figure 2(b)).

1) For each build, we identify what change sets were contained in the build. This information is available as an explicit link in the Jazz database.
2) We determine authorship for each file. Since each change set has only one author, we know who changed the files in each change set and in turn who changed each file. In weighted congruence, if an author modifies the same file in different change sets, then we add an additional edge for each time the author changed the file.
3) We determine dependencies. Because a file can exist in multiple change sets, we determine that if a file appears in multiple change sets, then there is a dependency between the files. In weighted congruence, an edge is added for each pair that appears in the build.
4) We calculate the coordination needs as per the Coordination Needs calculation in Section 5.2. The diagonals are ignored.
5) When applying weighted congruence, we normalize the technical dependency network by taking the highest-ranked edge and dividing every edge in the network by that value. This converts each edge to a value between 0 and 1, indicating the relative rank of a coordination need between two individuals in the network.

6.3.2 Conceptualizing Actual Coordination

We conceptualize actual coordination as communication through comments in the Jazz environment. A person is considered as coordinating with another person if both of them commented on the same work item. We calculate the actual coordination matrix as follows (Figure 2(c)).

1) We identify the comments from the work items involved in a build. Each build contains a number of change sets, and each change set has an associated work item with comments.
2) For each person who commented on a work item, add a communication edge to every other person who commented on the same work item. If calculating weighted congruence, there may be multiple edges between pairs of people because there may be multiple work items involved in a single build.
3) When applying weighted congruence, we normalize the communication network by taking the highest-ranked edge and dividing every edge in the network by that value. This converts each edge to a value between 0 and 1, indicating the relative rank of comment-based communication between two individuals in the network.

6.3.3 Statistical Methods

Most of our data is highly skewed to the left, meaning that we have a larger proportion of small values than we would expect from a normal distribution. We attempted to fit the data to a log approximation, but found that even with this transformation, the Kolmogorov-Smirnov Test showed significant differences between a normal distribution and our data for both unweighted and weighted congruence. To account for this we use non-parametric statistics to test significance between variables.

The non-parametric technique we use is the Wilcoxon rank-sum test (also known as the Mann-Whitney test) to test that two samples are drawn from the same population. In this paper, we consider a p-value of less than 0.05 as significant.

We use histograms to demonstrate distributions of the data. We use boxplots to show differences between data subsets especially between error builds and OK builds. A boxplot contains notches that illustrate where certain percentiles of data fall in. The thick notch is the median (50% percentile). The centre box's boundaries show the 75th percentile, and the 25th percentile values, meaning that 50% of the data points fall in the range of the box. The outer lines are the 95th percentile and the 5th percentile. Points that lie outside of these outermost lines are outliers.

We also isolate different factors into contingency tables and use the χ^2 test to check if the factors in the contingency table have an effect on our variable of interest.

6.3.4 Analysis Techniques

To answer Research Question 1, we conceptualize the congruence as described in Section 6.3, then separate the builds into error builds, and OK builds, and apply the Wilcoxon Rank-sum test on the congruence to determine if the two samples provided are drawn from the same population. We test the differences in congruence between the error builds and the OK builds.

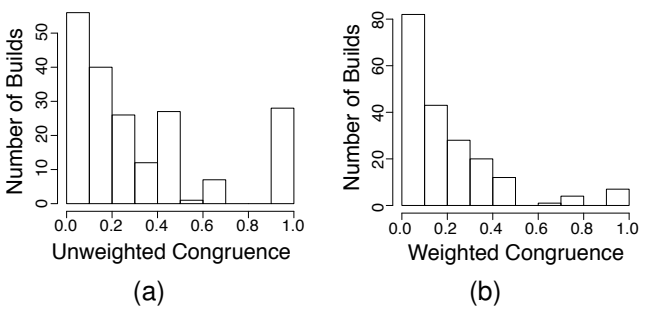


Fig. 3. Histograms of Unweighted Congruence (a) and Weighted Congruence (b) across Builds

To answer Research Question 2, we must use the weighted congruence measure. We conceptualize weighted congruence as described in Section 6.3. For each build, we calculate the gap size between each pair of people with a coordination need using the lack-of-coordination matrix explained in Section 5.5. Because of the normalization, each gap size is a value from 1 to -1, where 1 is a large gap with little or no communication, 0 is perfect congruence, and -1 is a gap where there is more communication than necessary to fulfill the coordination need. We take the mean of these gap sizes, and then compare the gap size mean in OK builds to the gap size mean in error builds using the Wilcoxon Rank-sum test.

To answer Research Question 3, we examine in detail overall characteristics of the Jazz project to explore other potential influences that may affect congruence, as well as build results. Data that we examine include the relationship between the number of work items in a build and the resulting build quality, as well as the type of build. We also discuss the effects of social factors, such as distance and awareness on congruence.

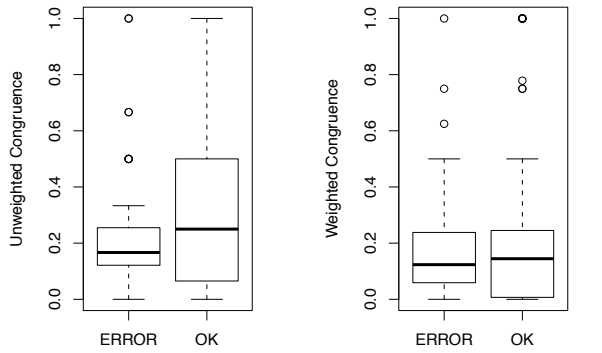
7 Results

In the Jazz repository, we analysed 197 builds; of these builds, 60 were error builds, and 137 were OK builds. The unweighted congruence has a mean value of 0.330, and the weighted measure has a mean value of 0.195, which are low values for congruence. Over 75% of the builds have a weighted congruence value of less than 0.25. Figure 3 shows the histogram of congruence values.

7.1 Effect of Congruence on Build Result

To answer Research Question 1, we test for a difference between the congruence index in error builds and the congruence index in OK builds. Figure 4 shows the boxplots for unweighted congruence and weighted congruence.

The result of a Wilcoxon rank-sum test indicates there is no significant difference between the congruence index of error builds and the congruence index of OK builds (W = 4460, p-value = 0.3414) (Figure 4(a)). Regardless of whether the congruence is high or low, the build result is unaffected.



(a) Unweighted Congruence (b) Weighted Congruence

Fig. 4. Boxplot of Comments and Coordination Needs per Build

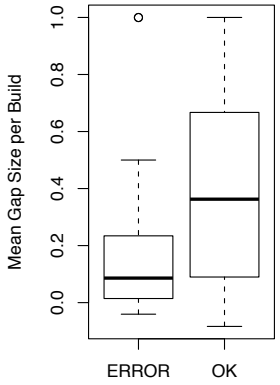


Fig. 5. Mean Gap Size per Build

We find that using weighted congruence has no effect on build result (W = 3658, p-value = 0.2178) either (Figure 4(b)).

7.2 Effect of Gap Size on Build Result

To answer Research Question 2, we use the Wilcoxon Rank-sum test to determine if there is a difference between the mean gap size in error builds, and the mean gap size in OK builds. Figure 5 shows the boxplots. Overall, the mean gap size ranges from -0.0384 to 1.0, with a mean of 0.317 and a median of 0.190. For error builds, the values range from -0.0405 to 1.0, with a mean of 0.147 and a median of 0.0860. For OK builds, the values range from -0.0834 to 1.0, with a mean of 0.391 and a median of 0.363.

The result indicated that there is a significant difference between the mean gap size for OK builds and the mean gap size for error builds (W=2238.5, p-value = 3.713e-07). Our result, however, reveals that the gap size for OK builds is higher than the gap size for error builds, which is the opposite of what we hypothesized (Figure 5).

7.3 Social and Technical Factors in Jazz that Affect Congruence And Build Quality

To answer our third research question, we took an opportunistic approach. We examined different social and technical factors that may affect congruence and build quality, in our attempt to find explanations for the lack of a well-defined relationship between the strength of socio-technical congruence and build quality in Jazz. Specifically, we started by examining characteristics of a build such as types of builds and the different levels of required coordination, the number of comments and coordination needs in a build, the number of work items in a build, the commenting behaviour by different people involved in a build's coordination, and the effect of all these factors on the build quality. Finally, we sought insights about the coordination involved in fully-congruent builds and incongruent builds.

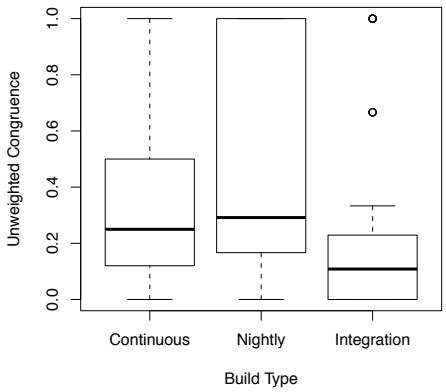


Fig. 6. Unweighted Congruence for types of builds

7.3.1 Effect of the Type of Build

In an interview with a Jazz project manager, the manager suggested that the type of build could have an influence on congruence. The Jazz team is most concerned with errors in continuous builds and nightly builds, and is only somewhat concerned about integration builds. The reason is that a continuous build involves a colocated team, and thus a low-congruence continuous build implies that the team is not coordinating well. As a consequence we decided to see if there are differences in congruence between the different types of builds.

We perform a pairwise Wilcoxon Rank-Sum test with the Holm p-value adjustment on three different types of builds: continuous, nightly, and integration builds (Figure 6). Our data contains 122 continuous builds, 55 nightly builds, and 14 integration builds. The test reveals that, unweighted congruence is different between continuous builds and integration builds (p-value = 0.0024) and nightly builds and integration builds (p-value = 0.0155). We found no difference between any build when using weighted congruence.

7.3.2 Communication and Dependencies

We narrow our investigation on the two elements that make up congruence: comments, and coordination needs. We present the frequency of the number of occurrences of communication and coordination for each build in Figure 7(a) and 7(b). The majority of builds

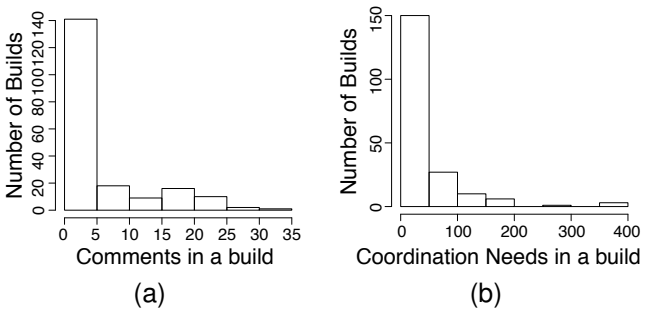


Fig. 7. Histograms of Comments (a) and Coordination Needs (b) across Builds

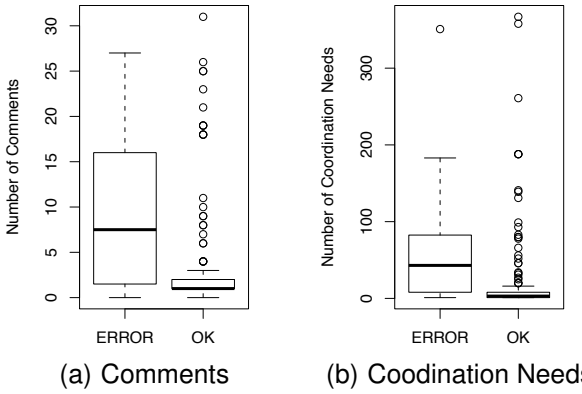


Fig. 8. Boxplot of Comments and Coordination Needs per Build

have a relatively small occurrence of comments and coordination instances.

We use the Wilcoxon Rank Sum test to determine if there was a difference in the number of comments in error builds, and in OK builds (Figure 8(a)). We find that there are significantly more comments in error builds compared to OK builds (W = 6214.5, p-value = 4.973e-09); error builds have 2.7 times more comments than OK builds.

Similarly, we use the Wilcoxon Rank Sum test to determine if there was a difference in the number of coordination needs, in error builds and in OK builds (Figure 8(b)). There are significantly more instances of communication needs in error builds compared to OK builds (W = 6362, p-value = 7.354e-10); error builds have 2.4 times more coordination needs than OK builds.

7.3.3 Work Items in Builds

Another influence on socio-technical congruence is the number of work items that a build is involved in. A work item is of interest in Jazz because it represents a unit of work. We would expect to see an inverse correlation between the number of work items and congruence on the rationale that the more work items that are involved in the build, the harder it is for people to coordinate.

To count the number of work items in a build, we identify the change sets in each build. Each change set is attached to a work item. Our results indicate that

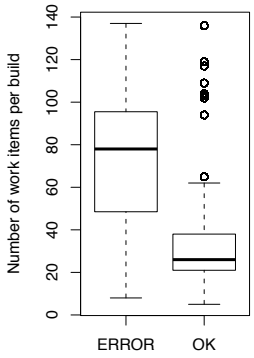


Fig. 9. Boxplot of Average Number of Work Items Referred to in OK builds and error builds

the number of work items range from 4 up to 182, with a mean value of 42.54 work items per build. The distribution is highly skewed to the left.

A correlation test using the Spearman ρ method on the number of work items per build versus unweighted congruence indicated that there was no correlation between the number of work items, and congruence (S = 2.04e+12, p-value = 0.7869, $\rho = -0.00178$).

The same correlation test on the number of work items per build versus weighted congruence indicated also that there is correlation between the number of work items and unweighted congruence (S = 1.88e+12, p-value $<$ 2.2e-16, $\rho = 0.0794$), but because the ρ value is so low, these results are not reliable.

We also investigate the relationship between the number of work items and build result. There is a statistically significant difference between the mean number of work items involved in an error build compared to an OK build (W = 61046557, p-value $<$ 2.2e-16) (Figure 9).

7.3.4 Commenting Behaviour by Change Set Authors

As congruence expects those who contribute technical work to a project to also communicate changes, we decided to examine commenting behaviour of change set contributors.

We examine whether or not the work items related to a change set have a corresponding comment posted by the contributor. We also identify if the work item is involved in an OK build or an error build. These results are shown in Table 1. A χ^2 test on this contingency table reveals no significant differences between the factors ($\chi^2 = 0.283$, df = 1, p-value=0.595). In 78% of these work items that appear in a build, the change set author also posts a comment on the associated work item.

7.3.5 Examining Extreme Congruence Values

We now come to a point where we are interested precisely in the differences between high-congruence builds and low-congruence builds. We further this investigation by looking at builds that have extreme values of congruence—0, where absolutely no coordination needs are satisfied with communication, and 1, where every

Result	Change sets without comment by contributor	Change sets with comment by contributor	Total
OK	1256	4622	5878
ERR	871	3295	4166
Total	2127	7917	10044

TABLE 1
Number of change sets in which a contributor commented in the corresponding work item

		Congruence	
	Result	1	0
Unweighted congruence	OK	26	31
	ERR	2	2
Weighted congruence	OK	6	31
	ERR	1	2

TABLE 2
Number of Builds with Extreme Congruence Values

coordination need is satisfied with communication. We chose to investigate the extreme cases to see if there were differences in the way people coordinated in perfect-congruence builds, and in incongruent builds. Table 2 shows the results.

In our data, 33 builds have both a weighted and unweighted congruence index of 0. Among these builds, 2 of them are error builds and 31 are OK builds. It appears that, even in a situation with with 0 congruence, the majority of these builds appeared to have build successfully.

We then examine the builds that have a congruence of 1, which suggests "perfect congruence". There are 28 builds that have an unweighted congruence index of 1; of these builds, 2 builds are error builds and 27 are OK builds. For weighted congruence, there are 7 builds that have a weighted congruence of 1. Of these 7 builds, 1 build is an error build and 6 are OK builds. The weighted congruence builds are subsets of the unweighted congruence builds.

We examined, specifically, the number of comments on these builds. Our results are shown in Table 3. Of note is the high number of comments on work items that have 0 congruence. This means that individuals who have no technical relationship to the work item are commenting.

We examined the work items with extreme amounts of congruence, reading the comments for any differences in the content discussed. Unfortunately, there were no discernible qualities between comments made in a build with a congruence of 0, and comments made in a build with a congruence of 1. In both builds, individuals discussed technical implementation details, provided updates to colleagues, or requested assistance from colleagues. Another observation made from reading the comments is that these work items for the cases where congruence is 0 or 1 do not appear to have strong dependencies on other work items.

		Congruence	
	Result	1	0
No comment	OK	273	12
	Error	63	42
Comment	OK	852	111
	Error	303	146
Total	OK	1125	123
	Error	366	188

TABLE 3
Number of work items containing comments

8 Discussion

Previous work on socio-technical congruence and the ideas shown in Conway's Law lead us to expect that team members must coordinate according to coordination needs suggested by technical dependencies in order to build software effectively. In this exploratory case study, we applied socio-technical congruence to study coordination and its relationship to build quality in Jazz. We also applied a new weighted congruence measurement to study how the size of a coordination gap affects build quality, and investigated in-depth what social and technical factors in Jazz affect congruence and build quality.

What we found was surprising: There was no relationship between high congruence and OK build results. We found that the average congruence was very low—only 20–30% of the coordination needs in the project were fulfilled with actual coordination. Even if there was absolutely no congruence, the build result was often an OK build.

This was not our only unintuitive finding. We also examined the socio-technical congruence gap size with our weighted congruence measure. A small gap size suggests that a majority of the coordination needs between two pairs of people are satisfied with actual coordination, whereas a large gap size suggests that only a small number of coordination needs between pairs of people are satisfied with actual coordination. We hypothesized that small gap sizes suggested good coordination and therefore would lead to OK build results. What we found was the opposite: that in fact small gap sizes occurred in error builds, and that large gap sizes occurred in OK builds.

Why do these unintuitive results occur? What occurs in the Jazz project that makes it appear that coordination, as reflected by socio-technical congruence, has no effect, or possibly harms build quality? We discuss the reasons below.

8.1 Jazz has Strong Awareness Mechanisms

Our first research question asked, Does socio-technical congruence have an effect on build quality in Jazz?

The answer is that there was no significant difference between the congruence in error builds and in OK builds using either unweighted congruence or weighted

congruence. Even when we compared builds that had extreme congruence values of 0 and 1, we observed that 31 of 33 builds with 0 congruence resulted in an OK build.

Second, the overall congruence for the majority of builds is low: over 75% of builds have a congruence of less than 0.25. Why is the congruence in Jazz so low, and why does it not affect the build results? Based on work item inspections and interviews with Jazz team members, Jazz developers coordinate their development efforts. However, the evidence suggests that team members in Jazz do not conduct all of their coordination through *explicit communication*. Rather, they use *implicit communication* that they get from the environment and from peers in order to address technical issues.

Jazz Support for Explicit Communication

The Jazz team members use the Jazz platform extensively to communicate with each other. Discussion with team members revealed that they do not use E-mail to discuss the Jazz project, and inspection of the mailing list reveals that its primary purpose is for announcements such as server outages rather than for collaborating work.

This leaves the Jazz work item comment system and instant messaging as avenues for communication, as well as internal face-to-face meetings. In a discussion with a Jazz product manager, we learned that while face-to-face interaction is efficient for solving local issues, it does not benefit remote teams, and the Jazz team as a whole encourages every team member to record discussions as comments.

Jazz as a Shared Workspace

The Jazz project is not typical of most industrial software projects, and shares many of its characteristics with open-source development. Their 47 teams are distributed across seven sites, and their "open commercial" development model centres around the use of the environment as a coordination mechanism. The public is also able to view and contribute to the Jazz project. This means that much of the work is centred around the Jazz system, which contains plans, source code, work items, and comments, as a "shared workspace" that every team member can view and modify.

The Jazz client software, therefore, helps a Jazz developer acquire and maintain *environmental awareness* of what is going on in the project, via the shared workspace. Jazz's awareness mechanisms feature a developer-centred dashboard that reports changes to the workspace, built-in traceability, user notifications, regularly-generated reports, and an optional web browser interface. For instance, when a change set is created, it is attached to a work item, thus ensuring that people who are involved with the work item receive notification that a new change set has arrived, and allowing individuals to monitor the work of other developers. This "automatic notification" system, which provides awareness through the workspace, cuts down the amount of explicit communication and allows people to coordinate implicitly.

Coordinating using the workspace is well-known in the computer-supported cooperative work domain [53]. Open-source developers in particular coordinate especially around source code [54] as well as mailing lists [26], [55] because there is little opportunity for face-to-face interaction.

In light of these results, we believe that the Jazz project team requires a congruence of only 0.2–0.3 for their tasks to be completed. Much of the need for explicit, point-to-point communication is mitigated by implicit communication and the use of the workspace to coordinate.

8.2 Communication Between Individuals Occurs When Problems Arise

Our second research question asked, Does gap size have an effect on build quality in Jazz? The gap size is our representation of whether enough communication occurred to fulfill multiple coordination needs. If a pair of developers has a lot of different dependencies with each other, one would expect them to coordinate more often as well.

In this case, the answer to our second research question is yes, but a qualification of this result is needed. We expected OK builds to have a small gap size, and that error builds would have a large gap size. Instead, we found that the mean gap size for error builds is 0.0860 and the mean gap size for OK builds is much larger at 0.363.

We saw evidence that the Jazz team is able to react to situations that involve multiple technical dependencies by communicating, a trait that has been previously observed among open-source developers [51]. Specifically, in our data, despite the fact that congruence was low for both OK and error builds, we saw that the number of comments in error builds was on average 2.7 times higher than in OK builds, and that the number of coordination needs in error builds was on average 2.4 times larger than in OK builds. One explanation is that the Jazz team is aware of builds that have high coordination needs, and are able to address the situation by coordinating with comments. If a work item was particularly complex during development, a developer will post comments on a work item asking for more information, informing others about unanticipated problems, and requesting expertise. This amount of communication does not occur in an OK build simply because the level of communication is not necessary. This would lead to the situation where builds with high coordination needs also have high communication.

8.3 Social and Technical Factors Affecting Congruence and Build Quality

Our third research question asked, What social and technical factors in Jazz affect congruence and build quality?

Our intention was to find factors that could explain the lack of relationship between socio-technical congruence and build quality, by investigating a number of factors that could possibly affect either the congruence or the build result. First, we identified that the type of build is related to the amount of necessary coordination and consequently affects congruence. Second, we identified that the number of work items in a build does not affect congruence because work items provide a location in the shared workspace for developers to comment, but they affect build quality. Finally, we identified that some features of the Jazz collaboration environment contribute to enhanced coordination behavior and that the communication of emergent members (those not captured in coordination needs matrices) may possibly contribute to the success of a build.

We found that the type of build does make a difference when it comes to the amount of coordination and consequently the socio-technical congruence values. Both continuous builds and nightly builds had similar levels of congruence, which is expected as these builds are done at the team level; these teams tend to be co-located. However, integration builds had significantly lower congruence values. Integration builds, rather than focusing on code changes, tend to focus instead on packaging issues. As each team in Jazz is responsible for a package that is integrated into the final product, full congruence is not necessary. Rather, integration can proceed by coordinating representatives for each package. If a particular package has problems, then the representative can go back to his development team and resolve the issue. It appears that high congruence is not necessary for integration builds.

Interestingly, the number of work items in a build did not have any correlation with unweighted congruence and the correlation with weighted congruence is inconclusive due to the weak reliability of the test. Yet, we observed that 78% of the developers that attach a change set to a work item also comment on that work item, implying that most developers who posts a change set also communicate their changes. This traceability encourages the change set author to communicate information about the change set, but does not necessarily require that other interdependent people coordinate as well. While this commenting behaviour increases the amount of communication in a build, it does not necessarily lead to increased actual coordination between those who have coordination needs, especially because comments are broadcasted.

These results illustrate two benefits of tracing work items to change sets.

First, connecting a work item closely to a change set allows a developer to communicate without the cognitive overhead of discovering who to communicate to. Developers are highly concerned about contacting the right people when they make modifications [32], but need to go through involved processes to find the appropriate contacts [24], [32].

Second, the openly-accessible work item comment system allows anyone, including people who are not in the coordination needs network, to contribute to the work item. We observed that many people who are not in a technical dependency with the change set also comment on the work item, providing an answer to a question or simply expertise on a topic. Jazz provides a way for a developer to write relevant information without needing to go through the cognitive effort of discovering who is affected by changes.

Other factors that influenced build quality included increasing number of work items involved in the build, increasing the number of comments on a build, and increasing the number of change sets in a build. None of these are necessarily surprising—increased complexity will contribute to failure. These findings align with related work on defect prediction [39], [56], [57]. These results illustrate the fact that congruence does not seem to prevent error builds and that simpler factors may be more accurate predictors of build failure.

8.4 Improvements to Socio-technical Congruence

Based on these results, we discuss how we can improve socio-technical congruence as an approach to measuring coordination.

In this paper, we presented a new measure that addressed a shortcoming of the existing unweighted congruence measure where there was no way to determine the strength of the relationship in coordination between two individuals. Despite the fact that our measure did not show that there was a statistically-significant result between congruence and build result, our new measure has provided additional explanatory power in Jazz by providing information about the gap size between two individuals. Thus, our measure provides additional explanatory power in a network without sacrificing any of the capabilities of unweighted congruence. Insights from gap size include the fact that as the number of coordination needs increase, the amount of communication increases as well, suggesting that developers are aware when a build incorporates many dependencies and react accordingly.

One useful further improvement to socio-technical measures is to improve the conceptualization of socio-technical congruence to incorporate indirect communication, especially the information that a developer receives from the environment. Software development centres around a workspace of code, work items, and computer-based documents, in addition to explicit communication.

A second improvement that becomes important from our study centres around identifying and incorporating people who were identified as being involved in actual coordination, but not in technical dependencies, into the congruence calculation. Such a person is an emergent individual [48]. In our study, we identified people who were coordinating with others even though they were not in the coordination needs matrix. This was illustrated

most clearly by the presence of comments on a build that has a congruence index of 0. One explanation for this result is that people other than the person who is technically related to the work item have information to share, or questions to ask about the work item. Current congruence conceptualizations ignore people who do not have coordination needs, but communicate in the actual coordination network.

This suggests that our conceptualization of exactly who should be related to a work item is not complete. We know that Jazz has a build coordinator that handles build issues. This person is not assigned to any change sets but is integral when solving issues involving builds or commenting on build status. Another person who may be involved in work items without having a technical dependency is the project manager. Two ways of incorporating this into the socio-technical congruence approach is to find a way to add the people we know observe project into the communication needs matrix as "monitors". Another approach is to introduce analysis that will identify which communications are emergent and which ones are expected as part of the coordination needs.

8.5 The Importance of Negative Results

It is a researcher's peril to discover that his or her hypothesis does not hold true. Traditionally, this simply means that the negative results are never reported. However, researchers in computer science, especially in empirical software engineering, realize the importance of negative results [58], [59]. A number of recent empirical software engineering papers [37], [40] have published negative results as a contribution, underlining their significance in empirical software engineering.

Specifically, our negative result is the finding that socio-technical congruence has no observable effect on build results in Jazz. This finding calls into question current conceptualizations of socio-technical congruence and suggests that more investigation is required in this area. We also present a new congruence measure that addresses a shortcoming of the existing congruence approach, and allows us to identify gaps, along with their strength. Although this study is only one study and the results cannot be generalized, we believe that highlighting issues with socio-technical congruence will lead toward new congruence conceptualizations as well as more exploration into its effects on software development.

9 Threats to Validity

As this study is an exploratory case study examining the relationship between congruence and build quality, there are a number of threats to validity.

One threat to validity is that our data is incomplete. Jazz does not keep a full archive of build results, and as a consequence, we do not have a full population from which to draw data from. However, we believe that the sample of 197 builds is a representative sample for our analysis because they are distributed evenly across the year.

As with many other studies that are focused on repository mining, we relied on commenting data in Jazz. We did not capture face-to-face communication nor did we capture personal messages. However, due to large geographical distances and time zone differences, the Jazz team's primary mode of collaboration is digital and in the work item comment system. We thus believe that conceptualizing actual coordination as comments reflects the majority of communication in Jazz.

We also observed some evidence in our study that the dependencies among change sets as well as the attached work items, may not be as strong as originally thought. This would suggest that not all dependencies would require coordination needs, as calculated by our congruence formula. As mentioned, we used the "files changed together" approach to analyse dependencies because previous results indicated that this approach was more robust than alternate representations of technical dependencies [3]. It is possible that this reliability is project-dependent however. Future work involves investigating change impacts in Jazz, and the effects of changes on congruence.

Finally, this study is one exploratory case study and the results are not generalizable, nor can they be directly compared to existing studies due to the differences in the projects under examination. We hope that future studies will be able to corroborate our results and contribute to empirical research in socio-technical congruence.

10 Conclusion

In this paper, we performed a study of coordination in Jazz using a socio-technical congruence approach. We applied an unweighted congruence measurement proposed previously in literature as well as a weighted congruence approach to the Jazz project in order to discover the effects of congruence on build results. However, we found no relationship between unweighted congruence and build results, nor weighted congruence and build results.

Our study however, has broader significance than the lack of relationship between socio-technical congruence and build quality in Jazz. We determined two main factors that reduce the need for high socio-technical congruence when current measures are used. First, Jazz is a specialized platform that enhances communication over distance and high awareness of work among developers. We believe that the Jazz platform may have had the effect of maintaining coordination, not only through explicit communication in comments, but also implicit communication through notifications and reports.

Second, we found that Jazz developers react to problem situations. If a build is easy to fix, then coordination is not strictly necessary—environmental awareness may be enough to allow the team members to coordinate.

However, if a work item is complex, then the developers communicate more often. Such complex work items lead to error builds.

Thus, we conclude that high socio-technical congruence may not always result in better build quality. This finding is significant because it provides empirical evidence that calls into question current conceptualizations of socio-technical congruence, and shows that increasing socio-technical congruence may not always lead to improvements in software development. Our future work will involve examining in more detail the relationships between particular individuals and builds, and to devise methods that incorporate emergent people and implicit communication into socio-technical congruence.

ACKNOWLEDGMENTS

The authors would like to thank the IBM Jazz Development Team for their cooperation and support of this research, and IBM Research for making the Jazz project data available. The authors would also like to thank the Natural Sciences and Engineering Research Council of Canada for funding this research.

REFERENCES

[1] 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 *CSCW '06: Proceedings of the 2006 conference on Computer-supported cooperative work, November 4–8*, 2006.

[2] K. Ehrlich, M. Helander, G. Valetto, S. Davies, and C. Williams, "An analysis of congruence gaps and their effect on distributed software development," in *MSR 2007: 4th Intl Conference on Mining Software Repositories, May 20–26*, 2007.

[3] 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 *Empirical Software Engineering Measurement (ESEM'08), Kaiserslautern, Germany*, 2008.

[4] I. Kwan, A. Schröter, and D. Damian, "A weighted congruence measure," in *Socio-Technical Congruence Workshop in conj. w/ Intl. Conf. on Software Engineering (ICSE '09), Vancouver, Canada*, 2009.

[5] S. Sawyer, "Software development teams," *Commun. ACM*, vol. 47, no. 12, pp. 95–99, 2004.

[6] C. R. B. de Souza and D. F. Redmiles, "An empirical study of software developers' management of dependecies and changes," in *International Conference on Software Engineering, Leipzig, Germany. May 10–18*, 2008.

[7] A. H. V. D. Ven, A. L. Delbecq, and J. Richard Koenig, "Determinants of coordination modes within organizations," *Americal Sociological Review*, vol. 41, no. 2, pp. 322–338, 1976. [Online]. Available: http://www.jstor.org/view/00031224/di974310/97p0268d/0

[8] R. Kraut and L. Streeter, "Coordination in software development," *Communications of the ACM*, vol. 38, no. 3, pp. 69–81, March 1995.

[9] J. Holck and N. Jørgensen, "Continuous integration and quality assurance: A case study of two open source projects," *Australasian Journal of Information Systems*, pp. 40–53, 2003/2004.

[10] D. Damian, L. Izquierdo, J. Singer, and I. Kwan, "Awareness in the wild: Why communication breakdowns occur," in *Second Intl Conference on Global Software Engineering, Münich, Germany. August 27–30*, August 2007, pp. 81–90.

[11] M. A. Cusumano and R. W. Selby, "How microsoft builds software," *Commun. ACM*, vol. 40, no. 6, pp. 53–61, 1997.

[12] J. D. Herbsleb, "Global software engineering: The future of socio-technical coordination," in *FOSE '07: 2007 Future of Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 188–198.

[13] A. Sarma, J. Herbsleb, and A. von der Hoek, "Challenges in measuring, understanding, and achieving socio-technical congruence," in *Socio-Technical Congruence Workshop held at the International Conference on Software Engineering (ICSE '08), Leipzig, Germany*, May 2008.

[14] P. Hinds and C. McGrath, "Structures that Work: Social Structure, Work Structure and Coordination Ease in Geographically Distributed Teams," in *Proceedings of the 20th Anniversary Conference on Computer Supported Cooperative Work*. New York, NY, USA: ACM, 2006, pp. 343–352.

[15] J. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," *Software Engineering, IEEE Transactions on*, vol. 29, no. 6, pp. 481–494, 2003. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1205177

[16] R. Frost, "Jazz and the eclipse way of collaboration," *IEEE Software*, vol. 24, no. 06, pp. 114–117, 2007.

[17] S. R. Fussell, R. E. Kraut, J. F. Lerch, W. L. Scherlis, M. M. Mcnally, and J. J. Cadiz, "Coordination, overload and team performance: effects of team communication strategies," in *CSCW '98: Proceedings of the 1998 ACM conference on Computer supported cooperative work*. New York, NY, USA: ACM Press, 1998, pp. 275–284. [Online]. Available: http://dx.doi.org/10.1145/289444.289502

[18] M. Conway, "How do committees invent," *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.

[19] M. E. Sosa, S. D. Eppinger, and C. M. Rowles, "The misalignment of product architecture and organizational structure in complex product development," *Management Science*, vol. 50, no. 12, pp. 1674–1689, 2004.

[20] G. Valetto, S. Chulani, and C. Williams, "Balancing the value and risk of socio-technical congruence," in *Socio-Technical Congruence Workshop (in conj. ICSE 2008)*, May 2008.

[21] S. Carter, J. Mankoff, and P. Goddi, "Building connections among loosely coupled groups: Hebb's rule at work," *Comput. Supported Coop. Work*, vol. 13, no. 3-4, pp. 305–327, 2004.

[22] S. Marczak, D. Damian, U. Stege, and A. Schröter, "Information brokers in requirement-dependent social networks," in *16th Intl Requirements Engineering Conference (RE08), Barcelona, Spain, Sept. 8–12*, 2008.

[23] K. Ehrlich and K. Chang, "Leveraging expertise in global software teams: Going outside boundaries," in *International Conference on Global Software Engineering 2006, Florianopolis, Brazil.*, 2006, pp. 149–158.

[24] K. Nakakoji, Y. Ye, and Y. Yamamoto, *Supporting Expertise Communication in Developer-Centered Collaborative Software Development Environments*, ser. Computer Science Editorial Series. Springer-Verlag, May 2010, ch. 11.

[25] J. A. Espinosa, S. A. Slaughter, R. E. Kraut, and J. D. Herbsleb, "Team knowledge and coordination in geographically distributed software development," *Journal of Management Information Systems*, vol. 24, no. 1, pp. 135–169, 2007.

[26] C. Gutwin, R. Penner, and K. Schneider, "Group awareness in distributed software development," in *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*. New York, NY, USA: ACM Press, 2004, pp. 72–81.

[27] S. Faraj and L. Sproull, "Coordinating Expertise in Software Development Teams," *MANAGEMENT SCIENCE*, vol. 46, no. 12, pp. 1554–1568, 2000. [Online]. Available: http://mansci.journal.informs.org/cgi/content/abstract/46/12/1554

[28] D. E. Perry, N. A. Staudenmayer, and L. G. Votta, "People, organizations, and process improvement," *IEEE Software*, vol. 11, no. 4, pp. 36–45, 1994.

[29] J. D. Herbsleb, H. Klein, G. M. Olson, H. Brunner, J. S. Olson, and J. Harding, "Object-oriented analysis and design in software project teams," *Human-Computer Interaction*, vol. 10, no. 2/3, pp. 249–293, 1995.

[30] D. Redmiles, A. van der Hoek, B. Al-Ani, S. Quirk, A. Sarma, R. Silva Filho, C. de Souza, and E. Trainer, "Continuous coordination: A new paradigm to support globally distributed software development projects," *Wirtschaftsinformatik*, vol. 49, pp. 28–38, 2007.

[31] J. D. Herbsleb and R. E. Grinter, "Architectures, coordination, and distance: Conway's law and beyond," *IEEE Software*, vol. 16, no. 5, pp. 63–70, 1999.

[32] C. R. B. de Souza and D. Redmiles, "The awareness network: To whom should i display my actions? and, whose actions should

i monitor?" in *European Conference on Computer Supported Cooperative Work, 24–28 September 2007, Limerick, Ireland*, L. Bannon, I. Wagner, C. Gutwin, R. Harper, and K. Schmidt, Eds., 2007.

[33] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in *ICSE '07: Proceedings of the 29th international conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 344–353.

[34] T. Wolf, A. Schröter, D. Damian, L. D. Panjer, and T. H. Nguyen, "Mining task-based social networks to explore collaboration in software teams," *IEEE Software*, vol. 26, no. 1, pp. 58–66, 2009.

[35] T. Wolf, A. Schröter, D. Damian, and T. Nguyen, "Predicting build failures using social network analysis on developer communication," in *ICSE '09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–11.

[36] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter, "An empirical study of global software development: Distance and speed," in *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*. IEEE Computer Society, 2001, pp. 81–90. [Online]. Available: http://portal.acm.org/citation.cfm?id=381481

[37] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does distributed development affect software quality? an empirical case study of windows vista," in *International Conference on Software Engineering (ICSE '09), Vancouver, Canada*, May 2009.

[38] N. Nagappan, B. Murphy, and V. Basili, "The Influence of Organizational Structure on Software Quality: An Empirical Case Study," in *Proceedings of the 30th International Conference on Software Engineering*. New York, NY, USA: ACM, 2008, pp. 521–530.

[39] M. Cataldo and S. Nambiar, "Quality in global software development projects: A closer look at the role of distribution," in *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, July 2009, pp. 163–172.

[40] T. Nguyen, T. Wolf, and D. Damian, "Global software development and delay: Does distance still matter?" *Global Software Engineering, International Conference on*, vol. 0, pp. 45–54, 2008.

[41] T. Niinimäki and C. Lassenius, "Experiences of instant messaging in global software development projects: A multiple case study," *Global Software Engineering, International Conference on*, vol. 0, pp. 55–64, 2008.

[42] D. Damian, I. Kwan, and S. Marczak, "Requirements-driven collaboration: Leveraging the invisible relationships between requirements and people," in *Collaborative Software Enginnering*, A. Frinkelstein, J. Grundy, A. van der Hoek, I. Mistrik, and J. Whitehead, Eds. Springer-Verlag, 2010, ch. 3.

[43] K. Ehrlich, M. Helander, G. Valetto, S. Davies, and C. Williams, "An Analysis of Congruence Gaps and Their Effect on Distributed Software Development," in *Proceedings of the First International Workshop on Socio-Technical Congruence*, 2008.

[44] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *Commun. ACM*, vol. 31, no. 11, pp. 1268–1287, 1988.

[45] C. R. B. d. de Souza, D. Redmiles, L.-T. Cheng, D. Millen, and J. Patterson, "How a good software practice thwarts collaboration: the multiple roles of apis in software development," in *SIGSOFT '04/FSE-12: Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*. New York, NY, USA: ACM Press, November 2004, pp. 221–230. [Online]. Available: http://portal.acm.org/citation.cfm?id=1029925

[46] E. Trainer, S. Quirk, C. de Souza, and D. Redmiles, "Bridging the gap between technical and social dependencies with ariadne," in *eclipse '05: Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*. New York, NY, USA: ACM, 2005, pp. 26–30.

[47] S. Marczak, I. Kwan, and D. Damian, "Investigating collaboration driven by requirements investigating collaboration driven by requirements in cross-functional software teams," in *Collaboration and Intercultural Issues on Requirements: Communication, Understanding and Softskills (CIRCUS), in conj. with Requirements Engineering 2009, August 31, Atlanta, Georgia, USA.*, 2009.

[48] D. Damian, S. Marczak, and I. Kwan, "Collaboration patterns and the impact of distance on awareness in requirements-centered social networks," in *15th Intl Requirements Engineering Conference, New Delhi, India. October*, 2007.

[49] L. Hossain, A. Wu, and K. K. S. Chung, "Actor centrality correlates to project based coordination," in *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, 2006.

[50] R. S. Burt, "Structural holes and good ideas," *American Journal of Sociology*, vol. 110, no. 2, pp. 349–399, September 2004.

[51] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 3, pp. 309–346, 2002.

[52] G. Valetto, M. Helander, K. Ehrlich, S. Chulani, M. Wegman, and C. Williams, "Using software repositories to investigate socio-technical congruence in development projects," in *Fourth Intl. Workshop on Mining Software Repositories (MSR'07)*, 2007.

[53] K. Schmidt and C. Simone, "Coordination mechanisms: Toward a conceptual foundation of cscw systems design," *Computer Supported Cooperative Work*, vol. 5, pp. 155–200, 1996.

[54] F. Bolici, J. Howison, and K. Crowston, "Coordination without discussion? socio-technical congruence and stigmergy in free and open source software projects," in *Socio-Technical Congruence Workshop held at the International Conference on Software Engineering (ICSE '09), Vancouver, Canada*, May 2009.

[55] A. Mockus, R. T. Fielding, and J. Herbsleb, "A case study of open source software development: the apache server," in *ICSE '00: Proceedings of the 22nd international conference on Software engineering*. New York, NY, USA: ACM Press, 2000, pp. 263–272.

[56] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *ICSE '05: Proceedings of the 27th international conference on Software engineering*. New York, NY, USA: ACM, 2005, pp. 284–292.

[57] P. Yu, T. Systa, and H. Muller, "Predicting fault-proneness using oo metrics: An industrial case study," *Software Maintenance and Reengineering, European Conference on*, vol. 0, p. 0099, 2002.

[58] L. Prechelt, "Why we need an explicit forum for negative results," *Journal of Universal Computer Science*, vol. 3, no. 9, pp. 1074–1083, September 1997.

[59] W. F. Tichy, "Hints for reviewing empirical work in software engineering," *Empirical Software Engineering*, vol. 5, no. 4, pp. 309–312, 12 2000. [Online]. Available: http://dx.doi.org/10.1023/A:1009844119158