Does Socio-Technical Congruence Have An Effect on Software Build Success? 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 conduct an exploratory case study of coordination in the IBM® Rational Team Concert® project, which consists of 151 developers over seven geographically distributed sites, and expect that high congruence leads to a high probability of successful builds. We examine this relationship by applying two congruence measurements: an existing unweighted congruence measure in the literature, and a weighted measure that overcomes limitations of the existing measure. We discover that there is a relationship between socio-technical congruence and build success probability, but only for certain build types, and observe that in some situations, higher congruence actually leads to lower build success rates. We also observe that a large proportion of zero-congruence builds are successful, and that socio-technical gaps in successful builds are larger than gaps in failed builds. Analysis of the social and technical aspects in IBM® Rational Team Concert® allows us to discuss the effects of congruence on build success. Our findings provide implications with respect to the limits of applicability of socio-technical congruence and suggest further improvements of socio-technical congruence to study coordination.

Index Terms—Empirical software engineering, socio-technical congruence, coordination, awareness, software quality, integration

1 Introduction

Coordinating the efforts of individuals working together in 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 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.

We describe a case study of socio-technical coordination and its effect on software builds in the IBM® Rational Team Concert® (RTC) software product[1]. 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** [**?**]. High socio-technical congruence has been shown to be a predictor of coordination success [**?**], [**?**]. The mismatches between the social and technical dimensions, or **gaps**, also have been observed to increase resolution times for software activities. The objective of this study is to investigate the effects of socio-technical congruence on high-coordination software development activities.

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

1. IBM, Rational, Jazz and Rational Team Concert are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

We seek to discover the relationship that congruence has on the probability that a regularly-scheduled software build will be successful. We conduct an exploratory case study of a large software project at IBM. 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 [**?**], and a weighted congruence approach that provides details about the size of a gap between a pair of individuals [**?**]. We also examine RTC's processes and tools to identify any explanations of the relationship between congruence and builds that we find.

Our results indicate that congruence affects build success probabilities for certain types of builds. In a continuous build, which involves code submitted by a colocated team, high congruence leads to a higher probability of build success. However, in an integration build, which involves code submitted by many distributed teams, high congruence actually reduces the probability of build success. We believe that this effect is due to socio-technical congruence having a stronger effect on colocated teams than on distributed teams. Although the congruence per build is relatively low at approximately 20% overall the team was able to coordinate to complete their builds. We also find that the mean size of a congruence gap correlates inversely with build success probability. Our findings on the socio-technical congruence and build quality in RTC are complemented by insights we obtained from other factors in RTC that influence both the socio-technical congruence and build success probability. Some of these counterintuitive results may

be explained also by the congruence conceptualizations that we use in our empirical study.

Our case study adds to the scarce but needed empirical evidence about socio-technical congruence measures in the investigation of coordination in software projects and allows us to discuss improvements to congruence conceptualizations and 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 **??**. We discuss related work in Section **??** and describe the research objective in Section **??**. We then describe in Section **??** the two measurements for socio-technical congruence that we used in our case study: an unweighted congruence measurement found in the literature [**?**], and a weighted congruence measurement from our own work [**?**]. We describe our research questions and methodology in Section **??**, and outline the results in Section **??**. Our discussion of these results in Section **??** provides an explanation of the effects we observed. We conclude the paper by outlining the threats to validity in our study in Section **??** and conclude in Section **??**.

2 Background and Motivation

Here we discuss coordination in software engineering and discuss socio-technical congruence as a way to measure coordination quality. We then present our case study of the IBM Rational Team Concert® project.

2.1 The Need for Coordination

Software is extremely complex because of the sheer number of dependencies [**?**]. Large software projects have a large number of components that interoperate with one another. The difficulty arises when changes must be made to the software, because a change in one component of the software often requires changes in dependent components [**?**]. 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" [**?**]. 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 [**?**].

A successful software build can be viewed as the outcome of good coordination because the build requires the correct compilation of multiple, dependent files of source code. A failed build, on the other hand, demotivates software developers [**?**], [**?**] and destabilizes the product [**?**]. While a failed build is not necessarily a disaster, it slows down work significantly while developers scramble to repair the issues. A build result thus serves as an indicator of the health of the software project up until that point in time.

Thus, a developer should coordinate closely with individuals whose technical dependencies affect his work in order to effectively build software. This brings forth the idea of aligning the technical structure and the social interactions [**?**], 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 in software engineering was brought to attention by Cataldo, et al. [**?**], though the concept has been explored in engineering [**?**] and management science [**?**]. 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 the technical entities. Socio-technical congruence states that if there is a coordination need between two people, these people should be coordinating. For example, 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 [**?**].

What socio-technical congruence offers is an approach to measure the coordination quality [**?**]. We can use this measurement to identify the effect of socio-technical congruence on software build quality.

2.3 IBM Rational Team Concert Case Study

IBM Rational Team Concert, or **RTC**, is an collaborative software development tool built on the Jazz™ architecture. It incorporates the integrated development environment, the source-code control system, the issue-tracking system, and collaborative development tools. The RTC team self-hosts, meaning that the team uses its own software to develop subsequent versions.

The following qualities make RTC an attractive case in which to study socio-technical coordination: (1) the distributed team, (2) the iterative process, and (3) the rich data repository. Distribution makes more difficult

to coordinate work effectively [?], [?] which RTC is supposed to mitigate by enhancing communication among project members [?]. This forces developers to explicitly record communication across sites using the product. RTC's iterative process encourages developers to "build early and often". In addition to supporting continuous builds, RTC does frequent integration builds, providing a large amount of data to analyse.

At the time of our study, the RTC development team involved a team distributed over 16 different sites located in the United States, Canada, and Europe. Seven sites were active in RTC development and testing. There were 151 active contributors at these locations. Each team, which was not necessarily confined to one geographical location, was responsible for developing a component of RTC. The team sizes range from 1 to 20 and had an average of 5.7 members. The number of developers per geographical site ranged from 7 to 24 and was 14.8 on average.

The project uses the *Eclipse Way* development process [?]. 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. RTC's process encourages frequent building, including continuous, nightly, weekly, and integration builds.

One unique aspect of RTC is its "open-commercial" development model. RTC is a commercial product developed by IBM, but has a publicly-accessible issue-tracking and reports system. This open-commercial model allows a large amount of information to flow into the team from the community. Communication between individuals is done through the issue-tracking comments, through instant messaging, through Internet Relay Chat, by phone, and face-to-face. 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 [?]. The original conceptualization of socio-technical congruence is in Conway's Law, which observed that product architecture reflects organizational structure [?]. Socio-technical congruence has been recognized as an important element of product design in the management sciences field [?]. 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 [?], [?]. Unfortunately there are only a handful of empirical research results that discuss the effects of the socio-technical congruence approach on software teams.

3.1 Coordination in Software Teams

Research in software-engineering coordination has examined interactions among software developers [?], [?], how they acquire knowledge [?], [?], and how they cope with issues including geographical separation [?], [?]. The ability to coordinate has been shown as an influential factor in customer satisfaction [?] and improves the capability to produce quality work [?].

Software developers spend much of their time communicating [?]. Because developers face problems when integrating different components from heterogeneous environments, developers engage in direct or indirect communication, either to coordinate their activities, or to acquire knowledge of a particular aspect of the software [?]. Herbsleb, et al. examined the influence of coordination on integrating software modules through interviews [?], and found that processes, as well as the willingness to communicate directly helped teams integrate software. De Souza, et al. [?] found that implicit communications are important to avoid collaboration breakdowns and delays. Ko, et al. [?] found that developers were identified as the main source of information about code issues. Wolf, et al. [?] used properties of social networks to predict the outcome of integrating the software parts within teams. This prior work establishes the fact that developers communicate heavily about technical matters.

Coordinating software teams becomes more difficult as the distance between people increases [?]. Studies of Microsoft [?], [?] show that distance between people that work together on a program determine the program's failure proneness. Differences in time zones can affect the number of defects in software projects [?].

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 early RTC development shows that the task completion time is not as strongly affected by distance as in previous studies [?]. Technology that has enabled distributed collaboration include topic recommendations [?] and instant messaging [?]. Processes are adapting to the fast pace of software development: the Eclipse way [?] emphasizes placing milestones at fixed intervals and community involvement.

3.2 Effects of Socio-technical Congruence

Current research suggests that attaining a high level of socio-technical congruence is beneficial to an organization. Evidence shows that higher congruence leads to faster completion of modification requests [?]. The presence of gaps increases the number of code changes [?], and a lack of coordination connections across system and organizational boundaries have a negative effect on performance [?].

Socio-technical gaps have been found to be an issue not only because they lower the congruence and

thus lowering productivity [?], but they become especially problematic in the context of distributed development [?]. Thus, researchers have proposed remedial actions when socio-technical congruence gaps are discovered [?]. Examples of actions include closing a gap by augmenting coordination and eliminating the gap by refactoring software.

The usefulness of socio-technical congruence depends on the conceptualizations of the social and the technical dimensions. Although communication is believed to help people coordinate it is not the only way to describe the social dimension. For instance, Cataldo et al. [?] 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. [?] 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 [?]. Gokpinar, et al [?] applied a congruence technique and discovered that a higher coordination deficit leads to a larger number of filed incident reports, implying reduced quality.

Socio-technical congruence has been explored outside of the software development field, particularly in engineering and management disciplines [?], [?], [?], [?]. Sosa [?] describes a formal technique to compute socio-technical congruence and identify "potentially unattended technical interactions", which are technical dependencies in modules that are not monitored. Gokpinar [?], independently of our work, developed a weighted socio-technical congruence technique that he applies to the automotive industry.

In summary, there is a need to coordinate effectively in software development across technical dependencies that may cross geographical boundaries, and socio-technical congruence is a useful mechanism for studying coordination.

4 Research Objective

Our research objective in this paper is to study the applicability of socio-technical congruence to a large, distributed software team using a socio-technical approach. The theory of socio-technical congruence suggests that increasing socio-technical congruence will improve the outcome of a software development activity that require a high amount of coordination.

We use a software build as our coordination-intensive software development activity and unit of success. A build result is "OK" when the software compiles with no errors and passes every test case. A build result is "error" when an error is discovered during compiling or testing. We know that builds are important to the RTC team: to them, build quality is almost code quality. An OK build implies that the team was able to achieve a successful outcome, and an error build implies that the team had a problem during development.

We conceptualize a coordination need between two individuals as a relationship indicating that these two individuals change the same file before a build. We conceptualize actual coordination between two individuals as an instance in which each individual writes at least one comment in at least one work item associated with either of the file changes.

We hypothesize that, when using congruence calculated by "files changed together" and "work item co-commenting", increasing congruence results in an increase in the probability of a successful build. We also anticipate that a small gap size increases the probability of a successful build.

Due to our conceptualizations, we expect that our calculated congruence values may be lower than what occurs in reality. First, our conceptualization will likely underestimate the actual coordination occurring in the project. Second, our dependencies may overestimate the dependencies between files in RTC. As RTC is a distributed project that follows a modular design, it is possible that many changes are not as strongly dependent as our conceptualization suggests. Nonetheless, in absence of more detailed data we believe that our conceptualizations capture the relationships between developers and illustrate potential dependencies where coordination issues may occur and contribute to a build failure.

We use two different congruence measurements to explore what advantages a new measure can provide. The first congruence measurement is an unweighted technique developed by Cataldo et al. [?]. The second congruence measurement is a weighted technique from our previous work [?], which offers additional features such as relationship strength between two people and variable size of a socio-technical congruence gap.

Research Question 1: Does socio-technical congruence have an effect on build success probability in RTC?

Research Question 2: Does gap size have an effect on build success probability in RTC?

Our work differs from previous socio-technical congruence studies because it applies two different congruence measures on software builds that occur every one to six weeks in a single project. This differs from previous studies that apply congruence to entire project releases. To our knowledge no application of weighted congruence has been applied to software engineering.

5 Calculating Congruence

We describe unweighted congruence as presented by Cataldo et al. [?] in Section **??**. Weighted congruence [?], described in Section **??** is a weighted congruence measure from our work that addresses limitations of unweighted congruence.

Box 1 Examples of technical dependencies

- Requirements that depend on each other [**?**], [**?**]
- Source code modules changed together in a change set [**?**], [**?**]
- Source code that has a call-graph dependency [**?**]
- Tasks that depend on other tasks [**?**]

Box 2 Examples of actual coordination

- Communication—A communicates with B [**?**], [**?**], [**?**], [**?**].
- Location—A is in the same location as B [**?**], [**?**].
- Team structure—A is in the same team as B [**?**].

5.1 Technical Entities and Social Relationships

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. Socio-technical congruence has focused on the source code file as the technical entity [**?**], [**?**], although work has also examined socio-technical congruence using a requirement [**?**], [**?**] or a task [**?**] as the technical entity. The choice of technical entity depends on the context of the study.

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 technical dependencies are in Box **??**.

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

Social relationships are identified through **actual coordination**, which indicates how people in the organization are actually coordinating. Note that, despite the nomenclature "coordination" used in previous work [**?**], 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 actual coordination appear in Box **??**.

5.2 Calculating Socio-Technical Congruence

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 is dependent on another technical 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 congruence on this edge. Given definitions

$$\text{Diff}(CN, AC) = card\{\text{diff}_{ij} | CN_{ij} > 0 \,\&\, AC_{ij} > 0\}$$

$$|CN| = card\{CN_{ij} > 0\}$$

the socio-technical congruence index is calculated as [**?**]:

$$\text{congruence} = \frac{\text{Diff}(CN, AC)}{|CN|}$$

where AC is an $m \times m$ matrix representing actual coordination. 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 Unweighted Socio-technical Congruence Calculation

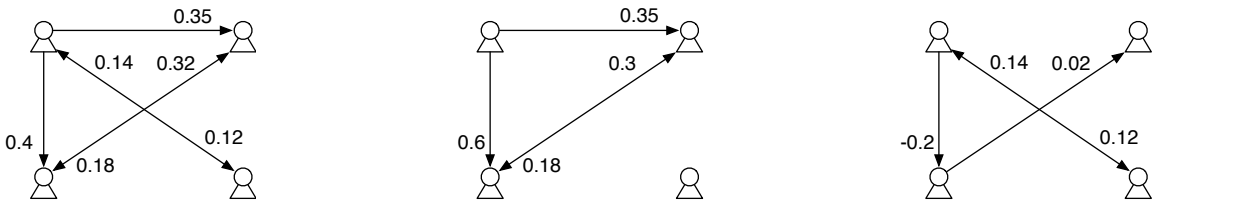
Unfortunately, the unweighted congruence measure has a limitation because socio-technical congruence identifies if a coordination need is either satisfied, or not satisfied, with no values indicating relative strengths.

The missing strengths prevent the identification of gaps that may be large or high-priority gaps that require attention. Previous research has shown that projects succeed despite the presence of gaps. Marczak, et al. [**?**] identified a project that delivered requirements on time despite the existence of gaps. Ehrlich, et al. [**?**] suggest that having an individual act as an intermediary between two groups, known as a *broker*, may be able to mitigate the effect of gaps. Communication structures may compensate for a lack of congruence [**?**]. Literature in areas such as management science supports the idea that gaps are not a liability, but simply a reality [**?**], [**?**].

As developers have multiple dependencies [**?**], [**?**], one would presume that those pairs with more coordination needs would need to coordinate more. Current socio-technical congruence provides no way to observe which pairs have high coordination needs. Researchers have suggested that both social and technical dependencies should be weighted [**?**], [**?**], but do not evaluate their proposed changes.

5.4 A Weighted Congruence Measurement

We present an extension to congruence based on prior work [**?**] that addresses the limitation discussed in the previous section. The intention is that this improvement will provide more information about coordination behaviour than the current congruence measure.



(a) The coordination needs (b) The actual coordination (c) The coordination that is lacking

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

5.4.1 Weighted Coordination Needs

This 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 illustrates the strength of the connection between a person i and an entity j. Such a connection may be the expertise person i has with a task j, or the priority that person i places on task 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. [**?**], 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', an $m \times m$ people by people matrix (Figure **??**). The coordination needs matrix tells you how strong the relations between people are supposed to be.

5.4.2 Weighted Actual Coordination

Weighted actual coordination is a relative value that indicates the strength of a relationship between two individuals (Figure **??**). 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

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 coordination matrix, but can also identify the size of these gaps (Figure **??**). This lack-of-coordination matrix can be used with both unweighted congruence and weighted congruence.

$$g_{ij}(CN'_{ij}, AC'_{ij}) = CN'_{ij} - AC'_{ij}$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, m$. 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 the value is positive, then there is not enough coordination to satisfy the information needs requested by the CN' matrix. If a value is negative, then there was more coordination than requested through the CN' matrix, and this particular connection is **overloaded**. Neither situation necessarily results in a problem, but we believe that investigating overload and lack of coordination has implications on software engineering 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, which is between 0 and 1, 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. 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. 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 an in-depth technique with which to study congruence.

5.8 Comparison with Other Weighted Measures

We know of three other methods for assigning a weight or importance to edges and gaps. Cataldo proposed a system using weights with integers [**?**]. Ehrlich, et al. [**?**] used a method to rank communication into three levels of importance. Gokpinar [**?**] independently proposed a weighted congruence measure.

5.8.1 Integer Matrices

Cataldo's method assigns integer weights as values in the task assignment and task dependency matrices. The existing formula in Section **??** considers that a single communication instance is sufficient for satisfying any amount of coordination. Our method provides information about overload and lack of coordination, and provides a more fine-grained representation through the the lack-of-coordination matrix.

5.8.2 Levels of Communication

Ehrlich's method aggregates multiple relationships between two individuals to determine a "level of communication". The relationships are: when two individuals work together on a project; when two individuals communicate several times a month for any reason; and when two individuals work together on shared files. If one or more of these relationships are true, they are added to provide a number representing the level.

The level of communication does not provide a ranking of gap size because they presume that a gap either exists, or that it is covered with one of the three levels. The authors mention that they rank gaps by size, but their gap rank is the total number of gaps that exist between two developers over the entire project. Thus, this measurement does not actually weigh the conceptual lack-of-communication distance between two individuals for each gap.

5.8.3 Coordination Deficit

Gopkinar independently proposed a weighted congruence called "coordination deficit", which is equivalent to our gap size. It weighs the number of dependencies between components, and the number of communications between individuals. Like our method presented in prior work [**?**], they calculate the coordination deficit by normalizing the edges and subtracting the values between the equivalent of an actual coordination network and a coordination needs network. The fact that the authors use normalization and algebraic subtraction suggests that they are reasonable approaches to calculating gap size.

6 Research Methodology

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

6.1 Research Questions

Research Question 1: Does socio-technical congruence have an effect on build success probability in the RTC project?

We hypothesize that high socio-technical congruence increases build success probability.

Research Question 2: Does gap size have an effect on build success probability in the RTC project?

We hypothesize that the mean gap size in a build is inversely correlated with build success probability. That is, a small mean gap size contributes is related to a larger probability of build success.

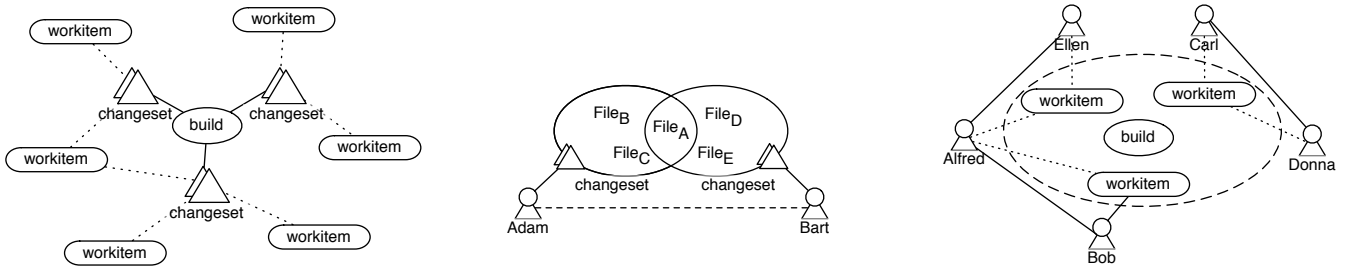
6.2 Data Description and Collection

In collaboration with IBM, we acquired a copy of the RTC repository containing information over a one year period. This period of time covers twelve "milestones", varying from 1 week to 4 weeks long, during which the RTC team achieves a number of objectives. The end of our data set coincides with a major milestone in which RTC was to be released for open beta testing.

In addition to quantitative data, we had obtained contextual information about the development team. We also investigated work items, comments, and reports available on the `jazz.net` web site.

The repository contains build information, work items, comments, change sets, and anonymised author information. We describe the data in more detail below.

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*. We extracted 2008 unique work items that were relevant to our study. The same work item may be involved in multiple builds; A total of 9218 non-unique work items were associated with a build across our data set. As our conceptualization



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

(b) Constructing a Coordination Needs Instance

(c) Constructing an Actual Coordination Instance

Fig. 2. Conceptualizing Socio-technical Congruence in IBM Rational Team Concert

focuses on the per-build level we examine the non-unique work items.

Build. A build is a compilation of the software to form a working unit. A build outcome can be an "error", which indicates that there was either a compilation error or a test suite error, "OK", which indicates no errors, and "Warning", which indicates that there were warnings returned by the compiler or the test suites.

A build contains the work from one or more work items, and these work items are not necessarily unique from build to build. The RTC repository does not keep a full record of every build over time, which means that we have more data points for the two most recent milestones compared to earlier in the project.

The RTC repository contains 533 builds, but we consider only builds that contain coordination needs, therefore dropping the number of builds to 214. We remove 17 builds whose result is Warning because the way that the RTC team treats its Warning builds is dependent on the build. Finally, we remove 7 builds whose types could not be identified. The resulting data set contains 191 builds.

There are three types of builds that we investigate in this study: *continuous builds*, *nightly builds*, and *integration builds*. Continuous builds, which are run regularly by a user, incorporates all changes from the local development site, plus known stable components from remote sites. Nightly builds also incorporate changes from the local site and stable components from remote sites approximately once a day. Integration builds integrate the latest components from every development site. The integration builds are often expected to break due to their large complexity, and that errors in continuous and nightly builds are potentially indicative of coordination problems. Our data contains 122 continuous builds, 55 nightly builds, and 14 integration builds.

Because we have a data set of builds over time, we include the date of a build as a confounding variable in our analysis. The build date is conceptualized as the number of days after the first build in our data set.

Comment. A comment is written text authored by a developer that is about a particular work item. Comments are the primary method of transferring information among developers in RTC. Multiple comments may be attached to a single work item. The data subset contains 9323 comments.

Change set. A change set is a collection of code changes in a number of files. When a developer updates the repository, all of the files that were changed together are updated in a single change set. A change set is generated by one author only, and is related to exactly one work item. A single work item may contain multiple change sets, and the same change set may occur in multiple builds. There are 3013 change sets in RTC. The number of change sets per work item ranges from 1 to 246. Ninety-five percent of the work items contain 15 or less change sets, and 42% of the work items contain only one change set.

Source code file. A file in RTC contains source code and are included in change sets. Over time, a file may be associated with multiple change sets.

Author. An author in RTC is someone who has contributed to RTC. In our conceptualization, we specifically identify an author as meaning someone who has submitted a change set containing modifications to files. Because each change set can be attributed to one author, there are always at least as many change sets as there are authors in a build.

6.2.1 Distribution of Build Data

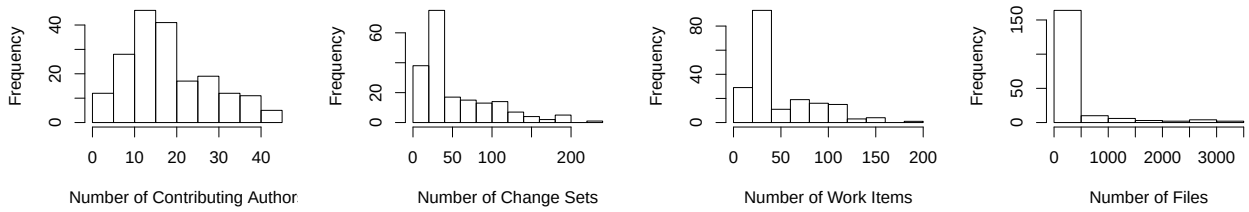
The 191 builds each involve a number of file authors, change sets, work items, and files. The same author, file, and work item may be involved across multiple builds. We plot the histograms for each of these entities in Figures **??**, **??**, **??**, and **??**. Most of these distributions, especially the files, are skewed toward the left.

6.3 Analysis Method

We use the two congruence measurements presented in Section **??** for our analysis: an unweighted congruence conceptualization, and a weighted congruence conceptualization. We explain how we conceptualize congruence in this section.

6.3.1 Conceptualizing Coordination Needs

Previous studies use various heuristics for determining the dependencies between modules. One conceptualization is based on method call graphs generated by static



(a) Authors per Build (b) Change Sets Per Build (c) Work Items Per Build (d) Files Per Build

Fig. 3. Distribution of authors, change sets, work items, and files per build

code analysis, and has been used by Ehrlich [**?**]. Another conceptualization is a "files-changed together" heuristic, used by Cataldo [**?**], where the files that were modified in the solution of a change request are considered to be dependent on each other. A third conceptualization is to use experts who manually identify dependencies [**?**].

We decided to construct dependencies based on a variation of the files-changed together heuristic because a "files-changed together" heuristic was more reflective of social relationships than a call-graph approach [**?**], .

We presume that a coordination need between two individuals exists if, minimally, these two individuals change the same file within the same build. In RTC, since only one author can be associated with each file check-in, there are two change sets involved, one associated with each file author. These two individuals should communicate with each other to ensure that the appropriate file dependencies are properly handled before the software is built (Figure **??**).

We generate the coordination needs matrix as follows.

1) For each build, we identify every change set included in the build. Every change set between the previous build and the current build is considered a part of the current build.

2) We determine authorship for each file to generate the assignment matrix A**.** Since each change set has only one author,the author is assigned to every file in that change set. If an author modifies the same file in a different change set, then add an additional edge for each change set in which the author changed that file.

3) We determine the file dependencies in a build to generate the dependency matrix D**.** We iterate through every change set in the build. Each file is dependent on every other file within a change set. Thus, if there are two change sets in build $_1$, C_1 containing A, B, and C, and C_2 containing A, D, and E, the resulting dependency matrix D would correspond to Figure **??**. In weighted congruence, add one additional edge for each file pair that exists in each additional change set in that build. Thus, in a build B_2 with change sets C_3={A, B, C} and C_4={A, B}, the dependency matrix entries in D would be $D_{AB} = 2$, $D_{AC} = 1$, and $D_{BC} = 1$.

4) We calculate the coordination needs as per the coordination needs calculation in Section ??. The diagonals are ignored. 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.

We use this conceptualization because of the explicit relationships between change sets, builds, and authors in RTC. In previous "files-changed-together" approaches, multiple files checked in together were associated with multiple authors, whereas in RTC, a change set is authored by one author only. Our conceptualization of coordination needs represents a situation in which two individuals work on a file that are in two different change sets, but are within the same build. Though the file-level changes may not necessarily be to the same lines of code, we presume that changes within a single file are sensitive enough such that a developer on that file benefits from being informed about them.

6.3.2 Conceptualizing Actual Coordination

We conceptualize actual coordination as communication through comments in the RTC environment. We treat those who comment on a work item as a group of communicators, and assign weights when individuals comment on multiple work items that others also comment on (Figure **??**).

Though the RTC commenting system allows users to post multiple posts to the same work item, we do not count multiple posts in a single work item as weighted communication. Since a work item comment can be read by anyone with access to the work item repository, and there is no threading in the work item comments, it is not reasonable to assume that there is point-to-point communication from one person directly to another. In addition, comment communication within the same work item do not correspond to weighted technical dependencies that cross different change sets.

We calculate the actual coordination matrix as follows.

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. For each work item, we include every comment that was posted after the previous build, but before the current build started.

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. In weighted congruence, add one additional communication edge for

each additional work item in which two individuals comment. We do not count weights for multiple comments in a single work item for the reasons discussed above. Thus, if work item W_1 contains commenters A, B, C and W_2 contains B, C, D, the resulting entries in the actual coordination matrix AC would be $AC_{AB} = 1$, $AC_{BC} = 2$, $AC_{CD} = 1$, $AC_{AC} = 1$, and $AC_{CD} = 1$.

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.

This conceptualization of weighted communication represents the number of people who post to work items in common, and aligns with the change sets.

We considered incorporating the RTC development mailing list into communication, but inspection of the mailing list revealed that they did not talk about technical issues and that the development list is primarily used for general announcements.

Unfortunately, due to the distribution of the 151 developers in the RTC team, as well as the one-year time period across which our study is conducted, we were unable to collect communication data such as instant messenger or face-to-face communication that we could associate to individual builds. Our concern with this missing data led us to inquire about the use of other forms of media among the team. We learned that although unofficial communication does happen between developers, the RTC team culture requires that the content of unrecorded and private communications be recreated as work item comments for consumption by remote project participants. However, we expect that socio-technical congruence will be lower than we expect as a consequence.

6.3.3 Conceptualizing Gap Size

The gap size in RTC represents a relative distance between the coordination needs and actual coordination. Since these two measures are not necessarily comparable to each other, we normalize both measures to identify a relative "rank" of coordination needs and actual coordination. For example, if there is a coordination need in a build that has many dependencies, intuitively we would want the relative amount of communication around that build to focus on that coordination need rather than a coordination need that has few dependencies. The gap size calculation identifies this mismatch.

We use the weighted congruence measure, as described in Section **??**, to compute the gap size. For each build, we calculate the gap size between each pair with a coordination need using the lack-of-coordination matrix explained in Section **??**. 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 full 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 use this as a variable in a logistic regression model. We also consider an alternative calculation of gap size where we do not punish extra communication and set all values of gap size less than 0 to 0.

6.3.4 Statistical Analysis Methods

To answer Research Question 1, we conceptualize congruence as described in Section **??** and test our hypotheses using logistic regression.

Logistic regression is ideal to test the relationship between multiple variables and a binary outcome, which in our study is a build result being either "OK" or "Error". The presence of many data entities in this project mean that we must consider confounding variables in addition to the socio-technical congruence when determining the effects on the probability of build success. Informally, logistic regression identifies the amount of "influence" that a variable has in the outcome of a build.

We show the relationship between a variable and the build success probability by plotting the y-axis as the probability. We use probability because we feel that it is more intuitive than odds ratios or logistic functions. If there is a relationship between a variable and the probability of build success, then we should see that as the variable's value increases, the probability also increases. In the probability figures, the solid line is the expected value, and the dashed lines indicate the 95% confidence intervals.

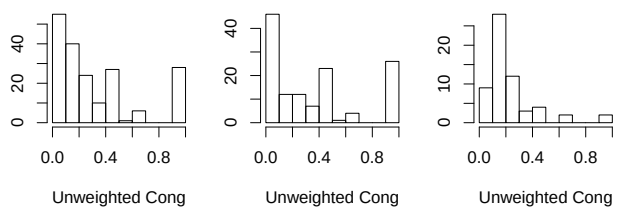
We run two different logistic regression models: one using weighted congruence, and one using unweighted congruence. We include the following variables: number of files per build, number of authors contributing to the build, number of files in the build, number of work items per build, the congruence, the build type, and the date of the build. We centre and scale each numeric variable. Because we were concerned about possible interactions affecting our results, we included first-order interaction effects and used backward stepwise elimination to remove variables to keep AIC (Akaike's Information Criterion) low.

To answer Research Question 2, we calculate the gap size and include this as a variable in our logistic regression model to study the effects on build success probability.

We use R [**?**] and the Design package [**?**] for logistic regression analysis.

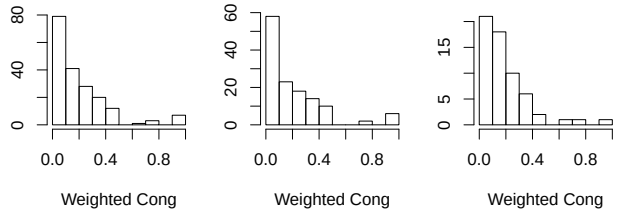
7 Results

In the RTC repository, we analysed 191 builds; of these builds, 60 were error builds, and 131 were OK builds. Table **??** displays summary statistics per build. Figure **??** displays histograms for unweighted congruence, and Figure **??** shows histograms for weighted congruence. The histograms compare the frequencies for each type of congruence for all builds, the OK builds, and the error builds only. There are some minor differences between



(a) All builds (b) OK builds (c) Error builds

Fig. 4. Distribution of Unweighted Congruence Values



(a) All builds (b) OK builds (c) Error builds

Fig. 5. Distribution of Weighted Congruence Values

	Min	Median	Max	Mean
Authors	2	17	44	18.62
Files	5	131	3101	342.3
Change Sets	4	34	226	54.2
Work items	4	34	182	48.3
Build date range (days)	0	345	361	319.2
Unweighted cong.	0	0.21	1	0.331
Weighted cong.	0	0.15	1	0.196
Gap size	-0.083	0.190	1.00	0.317

TABLE 1
Summary statistics

	2.	3.	4.	5.	6.	7.
weighted 1.	0.77	-0.22	-0.14	-0.12	0.05	0.02
unweighted 2.	–	-0.27	-0.22	-0.33	0.08	0.19
authors 3.		–	0.41	0.76	0.10	-0.30
files 4.			–	0.37	0.08	-0.20
changesets 5.				–	0.02	-0.38
workitems 6.					–	0.04
date 7.						–

TABLE 2
Pairwise Correlation of Variables per Build

unweighted and weighted congruence values; weighted congruence for instance largely reduces the number of "fully" congruent situations where congruence is 1.

The congruence values are low on average. The unweighted congruence has a mean value of 0.331, and the weighted measure has a mean value of 0.196, meaning that about one-third and one-fifth of the coordination needs are satisfied by actual coordination, respectively. Over 75% of the builds have a weighted congruence value of less than 0.25.

We calculated pairwise correlations between the variables weighted congruence, unweighted congruence, number of authors, number of files, number of change sets, number of workitems, and build date (Table **??**). There is a strong correlation between weighted and unweighted congruence, which is expected, a strong correlation between change sets and authors, and a weak correlation between authors and files. To avoid multicollinearity problems in our data, we choose to remove change sets from our logistic regression analysis because, due to the enforced processes in RTC, we know that there is exactly one author per change set, and thus there is at least as many change sets as authors per build. Note that weighted congruence and unweighted congruence do not appear in the same model and are thus not subject to multicollinearity problems.

To assess the fit of the logistic regression models, we use the Nagelkerke pseudo-R^2 and AIC. R^2 shows the proportion of variability explained by the model, and AIC is a measure of how well the model fits the data. Ideally, R^2 is high and AIC is low. Our current model contains 19 variables and has an R^2 of 0.579 and 0.548 for unweighted and weighted congruence models respectively. We compared our model in Table **??** to a model containing every first-order interaction effect with 27 variables and a model that contains the 7 main effects only (in Table **??**). We found that 19 variables is optimal and that removing further variables lowered the R^2 value while raising the AIC.

We observe also that the unweighted congruence model and the weighted congruence model are very similar, with a difference in Nagelkerke R^2 of only 0.030.

7.1 Effects of Congruence on Build Result

To answer Research Question 1, we perform a logistic regression analysis between the variables of the builds and the build outcome (Table **??**).

The result of logistic regression indicates that the following effects are significant for both unweighted and weighted congruence models: The congruence $\times$ build type effect, the congruence $\times$ build date interaction effect, the number of workitems $\times$ build date interaction effect, and the build date $\times$ build type effect. In addition, the number of authors and the number of files are significant main effects, although their coefficients are lower than the interaction effects involving congruence. We also identify unweighted congruence as a significant main effects in the unweighted congruence model.

In the next section we discuss the main effects and interactions effects that involve congruence affecting build probability. We discuss the effects of the non-congruence

		Unweighted		Weighted	
Model	Variables	AIC	R^2	AIC	R^2
Every interaction	27	188.6	0.595	196.7	0.559
Main effects only	7	213.2	0.269	214.2	0.263
Our model	**19**	**175.8**	**0.581**	**183.4**	**0.548**

TABLE 3
Model comparison

	Unweighted			Weighted		
Variable	Coef.	S.E.	p	Coef.	S.E.	p
Intercept	-0.5459	0.4663	0.2417	0.3289	0.4729	0.4867
congruence	**6.3410**	**1.6262**	****0.0001**	3.6699	2.3013	0.1108
authors	**-1.9759**	**0.5310**	****0.0002**	**-2.0176**	**0.5802**	****0.0005**
files	**-1.0734**	**0.4561**	***0.0186**	**-1.1169**	**0.5099**	***0.0285**
workitems	-0.1456	0.2355	0.5363	-0.1365	0.2343	0.5602
build type=I	**2.1533**	**1.0526**	***0.0408**	1.2777	1.0172	0.2091
build type=N	4.6833	200.7587	0.9814	2.7593	192.5236	0.9886
build date	-0.6560	0.6709	0.3282	-0.1352	0.7133	0.8497
congruence * build type=I	**-9.2151**	**2.5572**	****0.0003**	**-6.2748**	**2.9664**	***0.0344**
congruence * build type=N	-7.7308	91.8053	0.9329	-7.2024	188.1811	0.9695
congruence * build date	**-5.1266**	**1.9290**	****0.0079**	**-5.5670**	**1.9760**	****0.0048**
authors * build type=I	1.2688	0.7028	0.0710	1.1852	0.7370	0.1078
authors * build type=N	105.4123	535.8792	0.8441	103.0155	521.0808	0.8433
authors * build date	-0.6061	0.3616	0.0937	-0.6004	0.3576	0.0932
authors * files	0.7663	0.4289	0.0740	0.4979	0.5232	0.3414
files * build type=I	1.0920	1.1838	0.3563	1.7042	1.4099	0.2267
files * build type=N	-37.9274	199.2314	0.8490	-36.5156	192.3382	0.8494
workitems * build date	**0.8040**	**0.3003**	****0.0074**	**0.8909**	**0.3466**	***0.0102**
build type=I * build date	**2.6442**	**0.7678**	***0.0006**	**1.8627**	**0.7621**	***0.0145**
build type=N * build date	84.7252	344.8129	0.8059	84.3117	341.4988	0.8050
Model likelihood ratio	101.92	$p \approx 0$	$R^2 = 0.581$	94.36	$p \approx 0$	$R^2 = 0.548$
	191 observations			191 observations		

Build build type is set to continuous

$^*p < 0.05$; $^{**}p < 0.01$ Nagelkerke is used as the pseudo-R^2 measure

TABLE 4
Logistic Regression models predicting build success probability with main and interaction effects

	Unweighted			Weighted		
Variable	Coef.	S.E.	p	Coef.	S.E.	p
Intercept	0.5265	0.3040	0.0833	1.00416	0.2754	0.0003
congruence	0.9371	0.6807	0.1686	-0.85692	0.8544	0.3159
authors	**-0.5702**	**0.2003**	****0.0044**	**-0.64635**	**0.2023**	****0.0014**
files	**-0.6398**	**0.2477**	****0.0098**	**-0.70618**	**0.2571**	****0.0060**
workitems	-0.1755	0.1713	0.3055	-0.13229	0.1689	0.4335
build type=I	0.1693	0.4269	0.6917	0.06128	0.4154	0.8827
build type=N	0.2133	0.7791	0.7842	0.29418	0.7811	0.7065
build date	-0.1331	0.1821	0.4649	-0.11291	0.1819	0.5349
Model likelihood ratio	40.59	$p \approx 0$	$R^2 = 0.269$	39.52	$p \approx 0$	$R^2 = 0.263$
	191 observations			191 observations		

Build type is set to continuous

$^*p < 0.05$; $^{**}p < 0.01$ Nagelkerke is used as the pseudo-R^2 measure

TABLE 5
Logistic Regression models predicting build success probability with main effects only

effects, including the authors, files, workitems $\times$ date interaction effect and the date $\times$ nightly build effect in Section **??**.

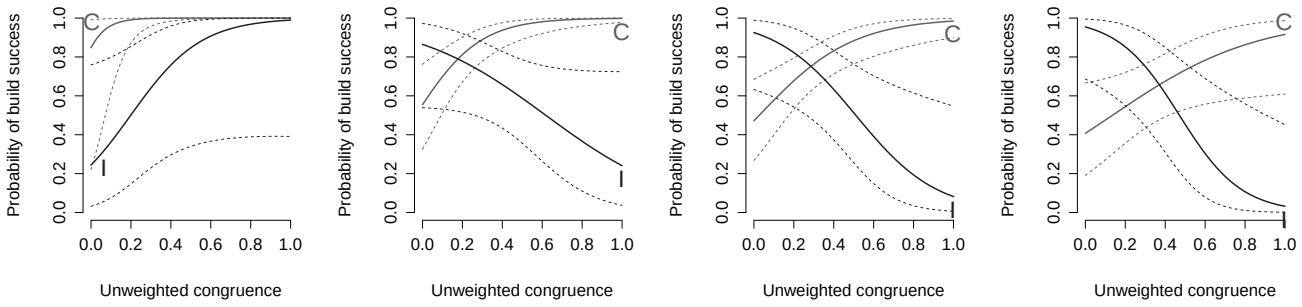
7.1.1 Effects of interactions involving congruence

The type $\times$ congruence interaction effect, the date $\times$ congruence interaction, and the type $\times$ date effect are each significant in our model (Table **??**). We plot in Figures **??** and **??** the effects of weighted congruence vs. probability of build success at the 10% date quantile (2008-01-25), at the 25% date quantile (2008-05-14), the 50% date quantile (2008-06-07), and the latest build (2008-06-26).

For continuous builds in weighted congruence (Figures **??**, in grey), an increase in congruence correlates to build success probability. The build date as the project ages appears to decrease the probability of build success for high-congruence builds more than low-congruence builds (Figure **??**). Note that, in the unweighted congruence model (Table **??**) the effect of unweighted congruence on continuous builds is significant, and that increasing congruence also increases the probability that a continuous build will succeed. However, the effect of weighted congruence in continuous builds is not as large as in unweighted congruence, nor is it considered statistically significant.

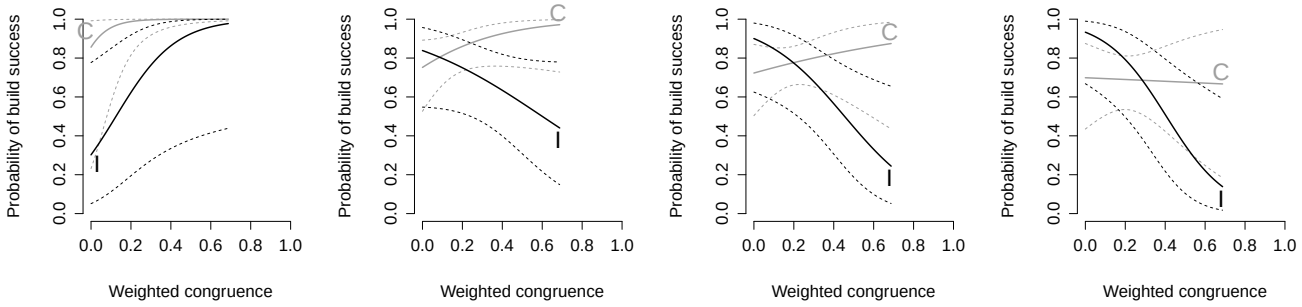
For integration builds (Figures **??**, in black), an increase in congruence decreases build success, with the exception of the 2008-01-25 build (Figure **??**). In in our 2008-01-25 build, we see that low congruence leads to low build probability, but high congruence has high build probability. As the project ages, this trend reverses and congruence is clearly inversely related with build success probability (Figure **??**).

The effect of congruence is totally opposite for contin-



(a) 2008-01-25 (b) 2008-05-14 (c) 2008-06-07 (d) 2008-06-26

Fig. 6. Estimated probability of build success for *unweighted congruence* and *continuous builds C* or *integration builds I* over time, adjusted to authors ≈ -0.156 (17 authors), files ≈ -0.352 (131 files), workitems ≈ -0.399 (34 work items)



(a) 2008-01-25 (b) 2008-05-14 (c) 2008-06-07 (d) 2008-06-26

Fig. 7. Estimated probability of build success for *weighted congruence* and *continuous builds C* or *integration builds I* over time, adjusted to authors ≈ -0.156 (17 authors), files ≈ -0.352 (131 files), workitems ≈ -0.399 (34 work items)

uous builds and integration builds in both unweighted and weighted congruence. Based on Figure **??**, increasing unweighted congruence significantly improve the continuous build success rate. However, increasing both types of congruence significantly decreases the integration build success rate.

7.2 Effect of Gap Size on Build Result

To answer Research Question 2, we fit a logistic regression model to determine the effect of the mean gap size on the build success probability. The mean gap size appears in Table **??** and Figure **??**.

We build logistic regression models based on the model in Table **??** using the gap size measurement. In the interest of saving space, we report only the odds ratio. We retain every significant interaction from our previous weighted congruence logistic regression in Table **??**. We compare two different models: Model G1 contains the gap size with weighted congruence as a variable, and Model G2 contains the gap size without weighted congruence. We do not use unweighted congruence as it is not possible to calculate gap size with unweighted congruence.

The effect of gap size on build result is significant in both models (Table **??**). This indicates that increasing the gap size significantly increases the odds that an OK build will occur, which is the opposite of what we hypothesized (Figure **??**). This means that if the gap size is large, the build success probability increases.

	Model G1	Model G2
Intercept	0.95	1.32
authors	0.43	0.60
files	0.63	0.63
workitems	0.75	0.85
build type=I	4.22	1.31
weighted cong	8.41	-
gapsize	7.71	8.71
builddate	1.81	0.59
authors * build date	0.40	0.74
workitems * build date	2.75	1.83
build type=I * build date	3.54	2.52
build type=I * weighted cong	0.01	-
weighted cong * build date	0.00	-

TABLE 6
Odds Ratio for Gapsize Models

7.3 Social and Technical Factors in RTC Affecting Build Success and Congruence

In light of our results, we examine not only the number of workitems × date significant interaction found in Section **??**, but different social and technical factors that may affect congruence and build success probability, in our attempt to find explanations for the lack of a well-defined relationship between socio-technical congruence and build success probability in RTC. Specifically, we

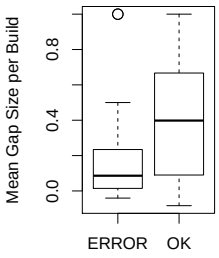


Fig. 8. Mean Gap Size per Build

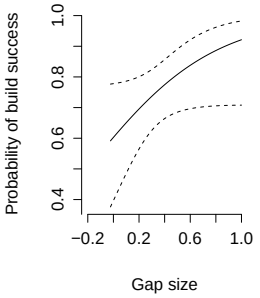


Fig. 9. Effect of gap size on build success probability, model G1.

examine the effect of build date on work items, coordination around fully-congruent builds and incongruent builds, and the effects of commenting behaviour on builds.

7.3.1 Other Effects on Build Success

Authors For both weighted and unweighted congruence, as the number of authors involved in a build increases, the probability that the build succeeds will decrease. The build probability is significantly lowered after more than 15 authors are involved in the build (Figure **??**). When over 30 authors are involved in the build, the estimated build success probability falls under 10%.

Files For both weighted and unweighted congruence, as the number of files involved in a build increases, the probability that the build will succeed will decrease (Figure **??**).

Build Date and Work items The work items $\times$ date interaction is significant in both weighted and unweighted congruence. Early in the project, as the number of work items increases, the probability of build success decreases (Figure **??**). As the project ages, this trend reverses and as the number of work items increases, the probability of build success increases as well (Figure **??**). According to the coefficients in Table **??**, this effect on build success probability is not as strong as the authors main effect or the files main effect.

Result	Change sets w/o comment by contributor	Change sets w/ comment by contributor	Total
OK	1278 (14%)	3956 (43%)	5234 (57%)
ERR	908 (10%)	3076 (33%)	3984 (43%)
Total	2186 (24%)	7032 (76%)	9218 (100%)

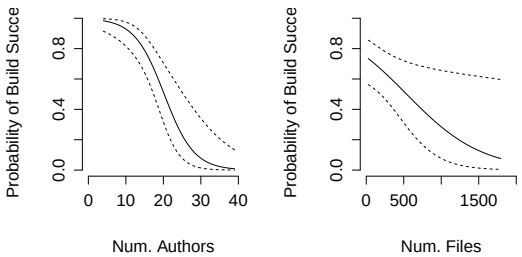
TABLE 7
Number of change sets in which a contributor commented in the corresponding work item

7.3.2 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 a work item related to a change set has a corresponding comment posted by the change set's author. We also identify if the work item/change set pair is involved in an OK build or an Error build. Note that a work item/change set pair can occur in multiple builds. We consider only comments posted before the build was started. These results are shown in Table **??**. The build result rates does not appear to be affected by whether the change set contributor commented: 43% of the builds resulted in error when the contributor commented, and 41% of builds resulted in error when the contributor did not comment. A χ^2 test on this table does not show a significant difference between the factors at the 0.05 threshold ($\chi^2 = 3.217$, df = 1, p=0.073). Overall, in 76% of the change set/work item pairs, the change set contributor also posted a comment in the associated work item.

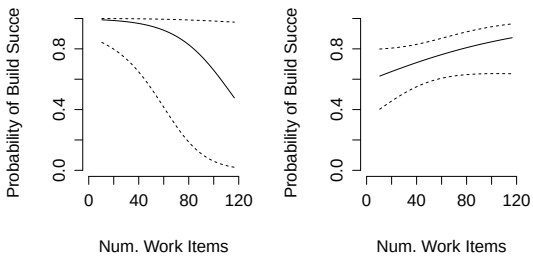
7.3.3 Examining Extreme Congruence Values

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 coordination need is satisfied with communication. We chose to investigate the extreme cases to see if there were differences in the way people



(a) Authors (b) Files

Fig. 10. Estimated probability of build success for *authors* and *files*, weighted congruence. Adjusted to workitems $\approx$ -0.399 (34), authors $\approx$ -0.156 (17), files $\approx$ -0.352 (131), w. congruence $\approx$ 0.1446, type = cont, date=2008-06-26



(a) 2008-01-25 (b) 2008-06-26

Fig. 11. Estimated probability of build success for *work items* and *date*, weighted congruence. Adjusted to authors $\approx$ -0.156 (17), files $\approx$ -0.352 (131), w. congruence $\approx$ 0.1446, type = cont

		Congruence	
		1	0
Unweighted	OK	26	30
	ERR	2	2
Weighted	OK	6	30
	ERR	1	2

TABLE 8
Number of Builds with Congruence Values 0 and 1

Congruence		Pairs w/Comments 1	Pairs w/Comments 0	Success rate 1	Success rate 0
No comment	OK	42	143	49%	69%
	Error	43	64		
Comment	OK	610	445	68%	69%
	Error	290	199		
Total	OK	652	588	66%	69%
	Error	333	263		

TABLE 9
Number of work items–change set pairs with comments and build success probabilities for congruence 0 and 1

coordinated in fully-congruent builds, and in incongruent builds. Table **??** shows the number of OK and error builds that occurred when congruence was equal to 1, and equal to 0. The weighted builds with full congruence are a subset of the unweighted builds with full congruence.

To determine if the presence of commenting affected the builds, we examined the number of comments on work item–change set pairs in builds with extreme unweighted congruence values. Our results are shown in Table **??**. build success probabilities improve when working on builds that have no comments, but of all work items involved in builds, ones with no comments are in the minority.

Of note is the high number of comments on work items that have 0 congruence. This indicates that individuals who have no technical relationship to the work item are commenting on the work item.

We manually inspected 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. We are unable to discover root causes of failure without a deeper examination of the technical changes and more knowledge of the RTC context.

8 Discussion

The concepts illustrated in Conway's Law, as well as previous empirical work on socio-technical congruence 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 success probability in RTC. We applied a modified weighted congruence measurement to study also how the size of a coordination gap affects build success probability, and investigated what social and technical factors in RTC affect congruence and builds.

Overall, we found that the average congruence across builds was very low—only 20–30% of the coordination needs in the project were fulfilled with actual coordination. Even in the cases where there is zero congruence, the build result was an OK build in over 90% of the observed cases.

Our first research question asked: **Does socio-technical congruence have an effect on build success probability in RTC?** We found that there was an interaction effect involving congruence and build type on build success probabilities (Section **??**). For continuous builds, increasing congruence improves the chance of build success slightly in continuous builds and can actually decrease build success probability in integration builds (Figures **??**). High unweighted congruence significantly improves continuous build success probability, and both unweighted and weighted congruence significantly reduce integration build success probability.

Our second research question asked: **Does gap size have an effect on build success probability in RTC?** The gap size is a representation of whether enough coordination occurred to fulfill multiple coordination needs. If two developers have multiple dependencies on each other, one would expect them to coordinate more often as well. We hypothesized that a small mean gap size would increase the probability of successful builds and that a large mean gap size would decrease the probability of a successful build. Instead, we found that as the mean gap size increases, the build success probability also increases (Figure **??**).

Below we discuss the reasons for these observed results based on our knowledge of the IBM Rational Team Concert project.

8.1 Strong Awareness Helps Coordination

The overall congruence for the majority of builds is low: over 75% of builds have a congruence of less than 0.25 (Section **??**). Despite low congruence, the RTC team is able to successfully build its software in many situations. The fact that the change set author commenting on a work item related to the change set does not appear to affect build results either (Section **??**), suggesting that changes do not need to be explicitly communicated for every build.

When we examined extreme congruence values, we observed 85% build success probability when weighted congruence is 1 and 93% build success when weighted congruence is 0 (Section **??**). If socio-technical congruence is a measure of coordination quality in software, and builds rely on coordination quality to be successful,

then there must be reasons why builds can succeed even when the congruence is zero.

First, because RTC is a highly-distributed project, the product under development uses a modular design [**?**] and thus is affected less by dependencies. Second, team members in RTC do not conduct all of their coordination through *explicit communication* even though work item inspection and discussion with developers indicate that the RTC corporate culture focuses on the work item as their base for communication. Rather, they use the *shared workspace* that incorporates cues from the environment and from peers in order to address technical issues. Both of these effects may contribute to congruence being lower than expected.

8.1.1 RTC supports Explicit Communication

The RTC team members use the RTC environment extensively to communicate with each other. RTC team members revealed that they rarely use private E-mail, and our 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 RTC work item comment system and instant messaging as methods for communication, as well as internal face-to-face meetings. We learned that while face-to-face interaction is efficient for solving local issues, it does not benefit remote teams, and the RTC team as a whole encourages every team member to record discussions as comments.

However, explicit coordination has a cost. There is evidence that involving too many authors in the same build also reduces the build success when using a weighted congruence conceptualization (Figure **??**). The overhead required to coordinate many people may interfere with the ability of the team to build the project successfully, suggesting that there is a limit before a developer is overloaded with information.

8.1.2 RTC is a Shared Workspace

The RTC client software helps a developer acquire and maintain *environmental awareness* of what is going on in the project, by providing access to a shared workspace. Much of the work is centred around the RTC technical entities, which include plans, source code, work items, and comments. RTC'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 of this change set. These "automatic notifications" cut 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 [**?**]. Open-source developers in particular coordinate around source code [**?**] as well as mailing lists [**?**], [**?**] because there is little opportunity for face-to-face interaction. RTC shares many characteristics with open-source development, such as a distributed team and a transparent development process.

In light of these results, we believe that, using our conceptualization, the RTC 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. We expect that the remaining congruence is covered through the RTC workspace, and through face-to-face communication, instant messenger, and phone communication. Though our congruence value appears low for the RTC team, the coordination in reality may be higher. Future studies should keep in mind that congruence may be lower than expected because of conceptualizations that cannot include every type of coordination in a project.

8.2 Coordination and Geographic Distribution

As RTC is a distributed team, geographic distribution has an effect on team performance, though the RTC environment helps mitigate some of these effects [**?**].

We learned that continuous and nightly builds involve mainly a colocated team, and that integration builds involve multiple components from multiple RTC teams in different locations. Our results indicate that congruence best benefits builds that occur within colocated teams.

It appears that, when communicating across distance, involving too many individuals when coordinating the activities of various teams may harm build success due to information overload [**?**]. To negate this effect, development leaders and build managers that have an overall view of the project are suited to coordinate teams to ensure build success [**?**].

8.3 Communication Between Individuals Occurs When Problems Arise

Our results indicate that even when congruence gaps are bridged, there is a high probability that a build still fails.

We have observed that, due to frequent awareness notices, the RTC team is able to react to situations with multiple technical dependencies by communicating quickly, a trait that has been previously observed among open-source developers [**?**]. Contributors are likely to comment on their own change sets (Section **??**).

The RTC team is aware of builds that require high coordination, 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 requesting expertise and informing others about unanticipated problems. This amount of written communication does not occur in an OK build simply because the level of coordination is not necessary; in essence, not every technical dependency requires explicit coordination to bridge that gap. There is also a possibility that developers coordinate on difficult

builds using methods that are not captured by our conceptualizations such as phone or face-to-face. More study of how developers coordinate under "problematic situations" is warranted.

8.4 Project Maturity and Build Success

We found that early builds exhibited a different type of relationship between congruence and build success probability than later builds (Section **??**). Over the course of the study, we observed 13 internal milestones; the last milestone in our observed builds was a public beta release for end users.

Build success probability decreased significantly over time for continuous builds and stayed roughly the same for integration builds (Figures **??**). However, the early builds in the project behaved contrary to later builds in the project (Figure **??** and **??**). The RTC software early in its lifetime is in a state of change. Integration builds are not a priority, and features are being added to the project. This means that dependencies are changing rapidly, as well as the expertise among team members, making it difficult to solidify coordination needs.

In addition to interactions between congruence and type, and congruence and date, we observed a significant interaction effect between build date and work items. We found that early in the project (Figure **??**), builds with large numbers of work items have a high probability of failing, but late in the project (Figure **??**), these builds succeed. Because the latest release was focused on a public release, a build linked to numerous work items may indicate that a bug or a feature is highly desired, and therefore received more attention. This is similar to the effect discussed with gap size (Section **??**), where an error build requires more coordination from involved developers.

8.5 Improvements to Socio-technical Congruence

Based on these results, we discuss our conceptualizations of congruence, and how we can improve socio-technical congruence as an approach to measuring coordination. This paper takes a step in the direction of exploring the situations of applicability that socio-technical congruence has to software projects. Our study is limited to digital, comment-based communication, but indicates the usefulness of socio-technical congruence in highlighting the relationship between communication, technical dependencies, and work outcomes.

We presented a weighted congruence measure that provides information which was not available in unweighted congruence, including gap sizes and relationship strengths. Although the weighted congruence measure yielded similar results to the unweighted congruence measure, the weighted congruence model appeared to lower the threshold of congruence by smoothing out the index values into a more balanced distribution compared to unweighted congruence (compare Figures **??** and **??**). This led to the unweighted congruence model having a significant main effect on continuous build success, whereas the weighted congruence model did not have such an effect. This shows that weighted congruence is a more conservative measure than unweighted congruence. Depending on the objective of the analysis, the unweighted congruence may represent an upper range of potential congruence in a project, whereas the weighted congruence tends to be less likely to report perfect congruence and may represent an average scenario.

However our conceptualizations of who should be related to a work item is not complete and can be improved. Many roles in the project are important, but do not have explicit technical dependencies. For example, we know that RTC has a build coordinator that handles build issues; This person is not assigned to any change sets. A person who is not identified as being part of the coordination needs network, but appears in the actual coordination network is called an *emergent person* [**?**]. 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 (Section **??**). Current congruence conceptualizations ignore people that do not have coordination needs but communicate in the actual coordination network. Thus one improvement is to incorporate people who are involved in actual coordination, but not in technical dependencies, into the congruence calculation.

A further improvement to socio-technical congruence measures is to improve the conceptualization of socio-technical congruence to incorporate indirect communication, especially the information that a developer receives from the environment. If environment information is transmitted through the software, then fine-grained logging will capture these automatic notifications and can be incorporated into congruence.

Another improvement would be modelling transitive communication because information often travels to individuals through other individuals. For example, an experienced development leader may receive information from multiple teams and forward information to appropriate individuals. Two individuals with a dependency may be able to satisfy their dependency who communicate through this development leader

8.6 Threats to Validity

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

Socio-technical congruence can be difficult to compare especially because conceptualizations such as technical dependencies and actual coordination vary from project to project. Some project-specific conceptualizations we used are the build as a measure of success, the fact that only files that are touched by more than one individual qualified as having dependencies on each other, and the work item comments connecting people in a clique.

Because of the context-sensitive nature of socio-technical congruence, especially with respect to the construction of communication and dependencies, it is difficult to apply socio-technical congruence as a benchmark for coordination. Having an understanding of the context of the project is extremely important when interpreting results obtained from socio-technical congruence effects.

Our coordination needs are likely overestimated due to the modular nature of RTC. As we are studying a distributed project that follows a transparent development process, RTC uses a modular design [**?**] and a change in a file in a change set may not necessarily be dependent on other changes within the same file. We observed evidence in our study that the dependencies among change sets as well as the attached work items may not be as strong as we have originally believed.

Our conceptualization of actual coordination underestimates the amount of communication that truly occurs in the project. As with many other studies that are focused on repository mining, we relied on commenting data in RTC. Due to geographical distance and time zone differences, the RTC team's primary mode of collaboration is the work item comment system. First, we assume that everyone involved in the work item reads every comment. Second, we do not take into consideration any additional coordination that may occur from a silent onlooker reading the comments. We believe intuitively that the first effect is greater than the second. This may help explain part of the effect of the unintuitive gap size result; rather than the gap being actually large in reality, it is large in our conceptualization because actual coordination outside of commenting is not recorded.

Another threat to validity is that our data does not cover every build executed in the lifetime of RTC. RTC 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. The threats are particularly high for early data points. The large confidence intervals in some of the builds, namely nightly builds, reflect this lack of data, thus we hesitate to draw conclusions based on early builds and nightly builds.

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. However, we believe that this study advances the theoretical and the empirical examination of socio-technical congruence.

9 Conclusion

In this paper, we performed a study of coordination in the IBM Rational Team Concert project using a socio-technical congruence approach. We applied an unweighted congruence measurement proposed previously in literature as well as a weighted congruence approach to RTC in order to discover the effects of congruence on build results. We found that both unweighted and weighted congruence have similar effects on build success probability. With respect to build success probability, we found an interaction effect between congruence and build results: if the build type was a continuous build, then increasing the congruence led to an increase in the build success probability. If the build type was an integration build, then increasing congruence actually led to a decrease in the build success probability. We also observed interaction effects between congruence and build date, the number of work items and build date, and build date and build type.

Our study however has broader significance than the relationship between socio-technical congruence and build success probability in RTC.

First, RTC is a specialized platform that enhances communication over distance and awareness of work among developers. Congruence among colocated developers increases build quality. We believe that the RTC platform may have had the effect of maintaining awareness among developers, not only through explicit communication in comments, but also implicit communication through notifications.

Second, we found that RTC 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 build and its work items are complex, then the developers communicate more often.

These findings are significant because they provides empirical evidence that illustrates the applicability of socio-technical congruence, and illustrates that an increase socio-technical congruence may not always be connected to improvements in software development, depending on the conceptualizations and context of the project. We believe that socio-technical congruence is extremely useful for studying the alignment of social and technical entities.

Acknowledgments

The authors would like to thank the IBM RTC Team, IBM Research, and the paper's anonymous reviewers. The authors thank NSERC 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 *Conf on Computer-supported Cooperative Work, Banff, Canada*, October 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 *Socio-Technical Congruence Workshop in conj. ICSE 2008, Leipzig, Germany*, 2008.

[3] I. Kwan, A. Schröter, and D. Damian, "A weighted congruence measure," in *Socio-Technical Congruence Workshop in conj. ICSE 2009, Vancouver, Canada*, 2009.

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

[5] 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 2008.

[6] 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.

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

[8] 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.

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

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

[11] J. D. Herbsleb, "Global software engineering: The future of socio-technical coordination," in *Future of Software Engineering in conj. ICSE 2007, Minneapolis, USA*, 2007, pp. 188–198.

[12] T. Browning, "Applying the design structure matrix to system decomposition and integration problems: a review and new directions," *Engineering Management, IEEE Transactions on*, vol. 48, no. 3, pp. 292 –306, August 2001.

[13] R. M. Henderson and K. B. Clark, "Architectural innovation: the reconfiguration of existing product technologies and the failure of established firms," *Administrative Science Quarterly*, vol. 35, pp. 9–30, March 1990.

[14] A. Sarma, J. Herbsleb, and A. von der Hoek, "Challenges in measuring, understanding, and achieving socio-technical congruence," in *Socio-Technical Congruence Workshop in conj. ICSE 2008, Leipzig, Germany*, May 2008.

[15] P. Hinds and C. McGrath, "Structures that Work: Social Structure, Work Structure and Coordination Ease in Geographically Distributed Teams," in *Conf on Computer Supported Cooperative Work*. New York, NY, USA: ACM, 2006, pp. 343–352.

[16] 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.

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

[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, Leipzig, Germany*, May 2008.

[21] S. Carter, J. Mankoff, and P. Goddi, "Building connections among loosely coupled groups: Hebb's rule at work," *Computer Supported Cooperative 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 *Intl Conf on Requirements Engineering, Barcelona, Spain*, September 2008.

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

[24] K. Nakakoji, Y. Ye, and Y. Yamamoto, *Supporting Expertise Communication in Developer-Centered Collaborative Software Development Environments*. Springer-Verlag, 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] S. Faraj and L. Sproull, "Coordinating Expertise in Software Development Teams," *Management Science*, vol. 46, no. 12, pp. 1554–1568, 2000.

[27] 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.

[28] 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.

[29] 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 Conf on Computer Supported Cooperative Work, Limerick, Ireland*, September 2007.

[30] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in *Intl Conf on Software Engineering, Minneapolis, USA*, 2007, pp. 344–353.

[31] T. Wolf, A. Schröter, D. Damian, and T. Nguyen, "Predicting build failures using social network analysis on developer communication," in *Intl Conf on Software Engineering, Vancouver, Canada*, May 2009, pp. 1–11.

[32] 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*, 2001, pp. 81–90.

[33] 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 *Intl Conf on Software Engineering, Vancouver, Canada*, May 2009, pp. 518–528.

[34] N. Nagappan, B. Murphy, and V. Basili, "The Influence of Organizational Structure on Software Quality: An Empirical Case Study," in *Intl Conf on Software Engineering, Leipzig, Germany*, May 2008, pp. 521–530.

[35] M. Cataldo and S. Nambiar, "Quality in Global Software Development Projects: A Closer Look at the Role of Distribution," in *Intl Conf on Global Software Engineering, Limerick, Ireland*, July 2009, pp. 163–172.

[36] T. Nguyen, T. Wolf, and D. Damian, "Global software development and delay: Does distance still matter?" in *Intl Conf on Global Software Engineering, Bangalore, India*, August 2008, pp. 45–54.

[37] T. Niinimäki and C. Lassenius, "Experiences of instant messaging in global software development projects: A multiple case study," *Intl Conf on Global Software Engineering, Bangalore, India*, pp. 55–64, July 2008.

[38] 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, Kaiserslautern, Germany*, 2008.

[39] 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," *Software Engineering Notes*, pp. 221–230, 2004.

[40] B. Gokpinar, W. J. Hopp, and S. M. R. Iravani, "The Impact of Misalignment of Organizational Structure and Product Architecture on Quality in Complex Product Development," *Management Science*, 2010. [Online]. Available: http://mansci.journal.informs.org/cgi/content/abstract/56/3/468

[41] M. Sosa, "A structured approach to predicting and managing technical interactions in software development," *Research in Engineering Design*, vol. 19, no. 1, pp. 47–70, 03 2008.

[42] D. Damian, I. Kwan, and S. Marczak, "Requirements-driven collaboration: Leveraging the invisible relationships between requirements and people," in *Collaborative Software Engineering*, J. Whitehead, I. Mistrík, J. Grundy, and A. van der Hoek, Eds. Springer-Verlag, 2010, ch. 3, pp. 57–76.

[43] 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. RE 2009, Atlanta, USA.*, August 2009.

[44] 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.

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

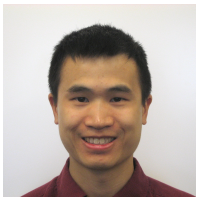
[46] L. Hossain, A. Wu, and K. K. S. Chung, "Actor centrality correlates to project based coordination," in *Computer-supported Cooperative Work, Banff, Canada*, 2006.

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

[48] 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 *Workshop on Mining Software Repositories in conj. ICSE 2007, Minneapolis, USA*, 2007.

[49] The R Project for Statistical Computing. [Online]. Available: http://www.r-project.org/

[50] Design: Design Package. [Online]. Available: http://cran.r-project.org/web/packages/Design/index.html
[51] A. MacCormack, J. Rusnak, and C. Y. Baldwin, "Exploring the structure of complex software designs: An empirical study of open source and proprietary code," *Management Science*, vol. 52, no. 7, pp. 1015–1030, July 2006.
[52] 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.
[53] 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 in conj ICSE 2009, Vancouver, Canada*, May 2009.
[54] C. Gutwin, R. Penner, and K. Schneider, "Group awareness in distributed software development," in *Conf on Computer-supported Cooperative Work, Chicago, USA*, November 2004, pp. 72–81.



Irwin Kwan Irwin is a Ph.D candidate at the University of Victoria with the Software Engineering Global interAction Lab (SEGAL). He received his M.Math from the University of Waterloo and a B.A.Sc in Software Engineering from the University of Ottawa. His interests are in empirical software engineering and requirements engineering, is particularly interested in studying collaboration patterns within software-development organizations.

Adrian Schröter Adrian is a Ph.D candidate at University of Victoria with the Software Engineering Global interAction Lab (SEGAL). He received his M.Sc. at Saarland University at the Software Engineering group of Prof. Andreas Zeller. His research interests cover Software Development in Distributed Teams, Software Quality and Mining Software Repositories.

Daniela Damian is an Associate Professor in the Department of Computer Science at the University of Victoria and the leader of the Software Engineering Global interAction Laboratory (SEGAL). Her interests are in collaborative software engineering with a special interest in global software development. She serves on the editorial boards of the Journals of Empirical Software Engineering, Requirements Engineering, and Software Process Improvement and Practice, as well as on program committees of conferences including Requirements Engineering, Software Engineering, Global Software Development and Foundations of Software Engineering.