Predicting Build Failures using Social Network Analysis on Developer Communication

Timo Wolf[1]
Siemens Corporate Technology
Germany
timowolf@uvic.ca

Adrian Schröter, Daniela Damian, Thanh Nguyen
Software Engineering Global interAction Lab (SEGAL)
University of Victoria, Canada
schadr@uvic.ca, danielad@cs.uvic.ca, duythanh@uvic.ca

Abstract

A critical factor in work group coordination, communication has been studied extensively. Yet, we are missing objective evidence of the relationship between successful coordination outcome and communication structures. Using data from IBM's Jazz™ project, we study communication structures of development teams with high coordination needs. We conceptualize coordination outcome by the result of their code integration build processes (successful or failed) and study team communication structures with social network measures.

Our results indicate that developer communication plays an important role in the quality of software integrations. Although we found that no individual measure could indicate whether a build will fail or succeed, we leveraged the combination of communication structure measures into a predictive model that indicates whether an integration will fail. When used for five project teams, our predictive model yielded recall values between 55% and 75%, and precision values between 50% to 76%.

1 Introduction

Communication problems lead to coordination and integration failures in work teams (e.g. [17, 21, 9]). This situation is further exacerbated in large and distributed teams, where effective communication and activity awareness of related but remote project members are problematic, yet key to anticipate and resolve coordination problems early (e.g. [17, 21]).

We strive to contribute to the growing body of research into the role of communication structures in determining coordination ease [24] and success [26]. Although there has been some research on the relationship between communication and coordination in work teams, research in software engineering is very limited. The study of the Enron email corpus found that central communicators exhibit a better ability to coordinate [26] and R&D teams with dense communication structures are associated with more coordination problems [24]. In open source software development, developers that are active in email communication have also been found to be most active in open source development [3]. In software engineering research, however it is unclear whether there are specific communication behaviors that enable effective coordination. Moreover, we are missing a precise conceptualization and objective measure of what successful communication in relation to project success is.

Complementary to previous research that is largely qualitative [22, 25] and which gathered information about occurrences of communication problems through project reviews and subjective ratings of project success, we use objective measures in studying the relationship between communication structures and coordination success. Past research has also largely investigated communication and coordination only in relation to entire projects. There is little systematic software engineering research in objectively examining the outcome of coordination (successful or failed) and assessing communication characteristics that lead to coordination failures. In this work we investigate the relationship between communication structures and coordination outcome at a finer level of detail. We study instances of coordination during the integration of code in large and distributed software teams, in relation to their associated communication structures. By communication structure we refer to the topology of the communication network that was involved in the tasks that lead to a software build. We use social network analysis measures such as density and centrality to obtain measurable characteristics of the communication structure.

Our study examines the data from IBM's distributed development project Jazz. Jazz is a development environment that focuses on collaboration support and tightly in-

[1]at the time of this study Timo Wolf was a PostDoc at the Software Engineering Global interAction Lab at the University of Victoria

tegrates programming, communication, and project management [12]. Our two step study revealed the following: Our results indicate that developer communication plays an important role in the quality of software integrations. Although we found that no individual measure could indicate whether a build will fail or succeed, we leveraged the combination of communication structure measures into a predictive model that indicates whether an integration will fail.

The remainder of the paper is structured as follows: we first discuss related work (Section 2) and introduce our two research questions (Section 3). Section 4 describes our methodology in the study of communication structures and integration in the Jazz project. Section 5 describes the results of our analysis. We close with their discussion and implications for practice in Section 7 and 8.

2 Communication, Coordination and Integration

The relationship between communication, coordination and project outcome has been studied for a long time in the area of computer-supported cooperative work. More recently the domain of software and distributed software development showed increased interest as well.

Communication plays an important role in work groups with high coordination needs and the quality of communication has been found as determinant of project success [6, 28]. The dynamic nature of work dependencies in software development makes collaboration highly volatile [5], consequently affecting a teams ability to effectively communicate and coordinate. Additional difficulties emerge in distributed teams, where team membership and work dependencies become even more invisible [7]. Moreover, team communication patterns are significantly affected by distance [24]. Maintaining awareness [37] becomes even more difficult when developers work in geographically remote environments; communication structures that include key contact people at each site are effective coordination strategies when maintaining personal cross-site relationships is challenging [24].

With respect to the role of effective coordination in project success, early studies indicate the issues that software development teams face in large projects [6]. A study by Herbsleb et al. [21] showed that Conway's law is also applicable for the coordination within development teams, supporting the influence of coordination on software projects. Kraut et al. [28] showed that software projects are greatly influenced by the quality of coordination of development teams. More recently a theory of coordination has been proposed and accounts for the influence of coordination on different project metrics such as rework and defects [23].

The importance of communication in successful coordination is also well documented and makes the study of communication structures important. For example, Fussell et al. [13] found that communication amount and tactics were linked to the ability of effectively coordinate in work groups. In software development, others showed that communication problems lead to problems during the activity of subsystem integration [17, 8]. Coordination conceptualized via communication has also been studied more generally in relation to project success: factors such as "harmony" [40], communication structure [33], and communication frequency [16] were related to project success.

In summary, the research so far suggests that coordination affects project outcome, but it leaves us without clear measurable evidence about this effect. The above studies, in software development or organizational behavior, have not used objective measures of communication or project success. Self-reported data was largely used in their analysis. Further, the relationship between coordination and project outcome was examined at a rather coarse-grain level – the project level – and by largely using only subjective measures for coordination and project success.

We analyze coordination at a finer level of detail, at the level of software subsystem integration and the integration of multiple subsystems and conceptualize the coordination success by the success of integration. In turn, we regard the integration to be successful if the associated software build, which includes compilation, testing and packaging, is successful. Thus, we are able to provide an objective measure of the coordination outcome.

The difficulty in studying failed integration in relation to communication lies in capturing and quantifying information about communication in teams that have a well-defined coordination goal but dynamic patterns of interaction. In our work we use the Jazz project data, which captures communication of project participants. This enables us to study the structure of the communication networks emerged around code integrations, both at individual teams of the project and within the entire project.

3 Can communication predict failure?

In order to examine the communication involved in the coordination necessary during subsystem integrations, we draw on social network analysis methods. Social network analysis has often been deployed to study communication networks of work teams. Using social network analysis has the major advantage that we can draw from its extensive knowledge of analysis and implications with respect to social, communication, and knowledge management processes [4, 11]. Griffin and Hauser [16] investigated social networks in manufacturing teams. They found that a higher connectivity between engineering and marketing increases the likelihood of a successful product. Similarly, Reagans and Zuckerman [35] related higher perceived outcomes to denser communication networks in a study of research and development teams.

Communication structure in particular – the topology of a communication network – has been studied in relation to coordination (e.g. [26, 24]) and a number of common measures of communication structure include network density, centrality and structural holes [41, 11].

Density, as a measure of the extent to which all members in a team are connected to one another, reflects the ability to distribute knowledge [36]. Density has been studied, for example, in relation to coordination ease [24], coordination capability [26] and enhanced group identification [35].

Centrality measures indicate importance or prominence of actors in a social network. The most commonly used centrality measures include degree and betweenness centrality having different social implication. Centrality measures have been used to characterize and compare different communication networks constructed from email correspondence of W3C (WWW consortium) collaborating working groups developing new technical standards and architectures for the web [14]. Similarly, Hossain et al. [26] explored the correlation between centrality in email-based communication networks and coordination, and found betweenness to be the best measure for coordination. *Betweenness* is a measure of the extent to which a team member is positioned on the shortest path in between other two members. People in between are considered to be "actors in the middle" and to have more "interpersonal influence" in the network(e.g. [14, 42, 26]).

The *structural holes* measures are concerned with the degree to which there are missing links in between nodes and with the notion of redundancy in networks [4]. At the node level, structural holes are gaps between nodes in a social network. At the network level, people on either side of the hole have access to different flows of information [18], indicating that there is a diversity of information flow in the network. Structural holes have been used to measure social capital in relation to the performance of academic collaborators (e.g. [15]).

Thus, having defined an objective measure of coordination outcome – the successful or failed integration build result (Section 2) – we investigate the role played by communication structure in software integration. Our first research question is:

RQ1: *Can individual measures of communication structure predict integration failure?*

Next, to further our investigation into the role played by communication in predicting integration failure, we go one step further and investigate whether the different communication structure measures can be combined into a prediction model that indicates whether an integration will fail.

Past research on failure prediction was not able to find a single code or code churn metric predicting failures [32, 1, 10], though the combination of those measurements became a strong predictor (e.g. [30]). This leads us to believe that even if we do not find a single communication structure measure that predicts integration outcome, it is useful to combine the communication network measures – as a reflection of the communication structure of a team – into a predictive model and study its predictive power.

Most prediction models in software engineering to date mainly leverage source code related data and focus on predicting failing software components or failure inducing changes (e.g. [2, 38, 42, 27]). Only few studies, such as Hassan and Zhang [19], stepped away from predicting component failures and used statistical classifiers to predict integration outcome. Recently, we can observe a trend towards leveraging developer networks, created upon code related dependencies, to predict component failures [34, 29]. In our work, we focus on the team coordination as given by their communication instead of source code, and similar to Hassan and Zhang predict integration outcome. Hence we state our second research question as follows:

RQ2: *Can the combination of communication structure measures predict integration failure?*

4 Methodology

To address our research questions we analyze data from a large software development project, IBM's Jazz [12]. With collaboration support as one of its main goals, Jazz provides integrated support for work planning and tracking, communication and collaboration, and continuous integration builds. For our research, Jazz provides the traceability between communication artifacts and development artifacts, important in the study of coordination processes. In what follows we first describe in detail the coordination and integration process in Jazz as the context for our investigation. We then explain how we conceptualized communication and integration outcome in Jazz, as well as the data collection instruments.

4.1 Coordination and integration in Jazz

The Jazz team is a large distributed team and uses the Jazz platform for development. The Jazz development involves distributed collaboration over 16 different sites located in the United States, Canada, and Europe. Seven sites are active in Jazz development and testing. There are 151 active contributors working in 47 teams at these locations, where contributors belong to multiple teams. Each team

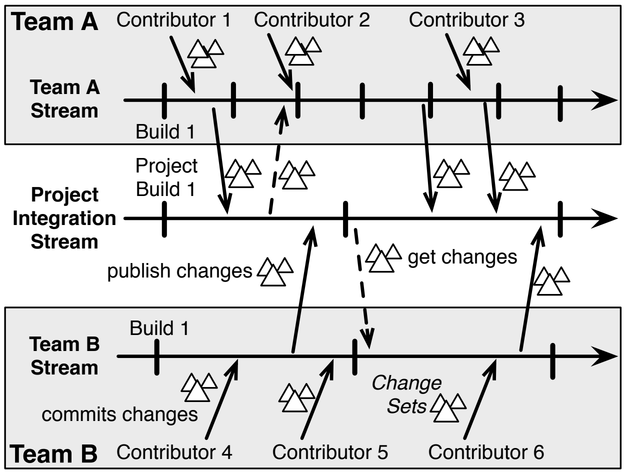

Figure 1. Teams contribute to their own source streams, which are then merged into one project stream.

is responsible for developing a subsystem or component of Jazz. The team size ranges from 1 to 20 and has an average of 5.7 members. The number of developers per geographical site ranges from 7 to 24 and is 14.8 in average.

The project uses the *Eclipse Way* development process [12]. 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.

As illustrated in Figure 1, the coordination process within each iteration requires the integration of subsystems developed by individual Jazz teams in a major milestone build of the product (referred to as beta build). Each team owns a source code *Stream* for collaboration and concurrent implementation of the subsystem. A Stream is the Jazz equivalent to a branch of a source configuration management system such as Subversion.

A continuous integration process takes place at team-level or project-level. In frequent intervals, each integration build (referred to a build henceforth) compiles, packages, and tests the source code of a stream. At the team-level, contributors commit code changes that are encapsulated in *Change Sets* from their own workspace to the *Team Stream*. The team integrations build the subsystem developed by the team. Once a team has a stable version within the Team Stream, the team publishes the change sets into the Jazz *Project Integration Stream* (see Figure 1). At the project level, the automated Jazz integration builds the subsystems of all teams. The Jazz project-level integration takes place *nightly*, *weekly*, and at the end of each iteration – *beta* build.

4.2 Coordination outcome measure

In our study we conceptualize the coordination outcome by the *Build Result*, which is regarded as a coordination success indicator in Jazz and can be `ERROR`, `WARNING` or `OK`. We analyze build results to examine the integration outcomes in relation to the communication necessary for the coordination of the build.

Conceptually, the `WARNING` and `OK` build results are treated similar by the Jazz team, as they require no further attention or reaction from the developers. In contrast, `ERROR` build results indicate serious problems such as compile errors or test failures and require further coordination, communication and development effort. We thus treated all `WARNING`s as `OK`s to clearly separate between failed and successful builds in our conceptualization of coordination outcome.

4.3 Communication in Jazz

Commenting on work items is the main task-related communication and collaboration channel used by Jazz. *Work Item*s represent single assignable and traceable tasks. A set of work items defines the tasks for each team in each iteration. Different types of work items represent defects, enhancements, and general tasks. The work items are assigned to and owned by contributors. The coordination necessary around the implementation of a work item is facilitated by contributors commenting or observing the communication around work items.

4.4 Communication measures

To investigate the communication of contributors involved in an integration build, we study the characteristics of the *communication network* for each build. In order to construct this network, we examine the content of the *Change Set*s associated with the build. Whenever an integration is executed, the change sets contain information on all changes made after the previous build, and the work items associated with the changes. By identifying the contributors who commented on these work items, we conceptualize communication among contributors coordinating for the build by their commenting behavior.

We use the example in Figure 2 to explain the construction of communication networks. For a $Build\ N$, the change sets made in the time range t_0 to t_1 are used to construct the $Build\ N\ Network$, which represents the communication that leads to $Build\ N$. To construct the *communication network* for a build we consider that:

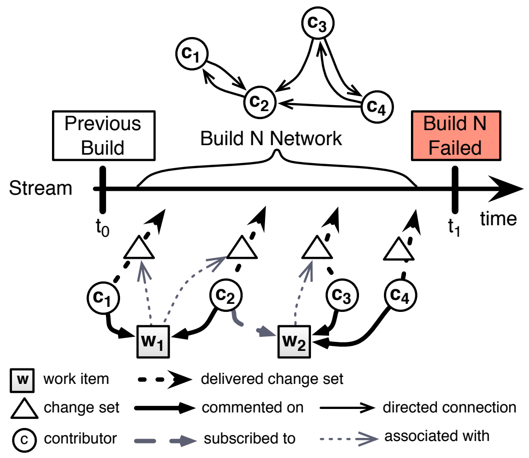

Figure 2. Communication networks construction example for a failed Build N.

1. the *nodes* represent the contributors involved in the build or its communication as follows:
   - (a) committers of change sets involved in the build, as they have a direct impact on the build result
   - (b) creators, commenters, and subscribers on the work items associated with change sets, as they communicated on the build related tasks
2. the *directed connections* represent communication flow from contributor c_i to c_j, communicating about a common set of work items that we name WIs if:
   - (a) c_i is creator of or commenter on WIs and provides information that is read by c_j
   - (b) c_j is commenter or subscriber of any work items WIs (assuming that c_j reads all comments of c_i)

For Build N in Figure 2 contributors c_1, c_2, c_3, and c_4 delivered change sets to the stream in the time range t_0 to t_1. Thus, they are added to the associated Build N Network. To connect the contributors in the network, we explore the work items w_1 and w_2 as they are associated with the delivered change sets. The contributors c_1 and c_2 commented on w_1. As we assume that c_2 reads all comments of the work items he comments on, we create a directed connection from c_1 to c_2, representing the communication flow. Vice versa, we create a directed connection from c_2 and c_1. As c_2 is only subscribed to work item w_2 and did not add any comments, we only create directed connections from c_3 and c_4 to c_2, as those contributors commented on w_2. The resulting communication-based social network Build N network represents the communication related to the development that leads to the failed $Build\ N$.

4.5 Communication network measures

To characterize the communication structure represented by the constructed networks for each build, we compute a number of social network measures. The measures that we include in our analysis are: Density, Centrality and Structural holes. Some of these measures characterize single nodes and their neighbours (ego networks), while others relate to complete networks. As we are interested in analysing the characteristics of complete communication networks associated to integration builds, we normalize and use appropriate formulas to measure the complete communication networks instead of measuring the individual nodes.

4.5.1 Density

Density is calculated as the percentage of the existing connections to all possible connections in the network. A fully connected network has a density of 1, while a network without any connections has the density of 0. For example, the density in the directed network in Figure 3 is $12/42 = 0.28$.

4.5.2 Centrality measures

We use the centrality measures *group degree centralization* and *group betweenness centralization* for complete networks, which are based on the ego network measures degree centrality and betweenness. The degree centrality measures for the ego networks are:

- The *Out-Degree* of a node c is the number of its outgoing connections $C_{oD}(c)$. E.g. $C_{oD}(c_1) = 2$ in Figure 3.
- The *In-Degree* of a node c is the number of it s incoming connections $C_{iD}(c)$. E.g.$C_{iD}(c_1) = 1$ in Figure 3.
- The *InOut-Degree* of a node c is the sum of its In-Degree and Out-Degree $C_{ioD}(c)$. E.g. $C_{ioD}(c_1) = 3$ in Figure 3.

To compute the *Group Degree Centralization* index for the complete network we use formula (1) from Freeman [11], in which g is the number of nodes in a network, and $C_D(c_i)$ is any of the degree centrality measures of a node c_i as described above. $C_D(c^*)$ is the largest node degree index for the set of contributors in the network. The formula is also used by [14, 24].

$$C_D = \frac{\sum_{i=1}^{g}[C_D(c^*) - C_D(c_i)]}{(g-1)^2} \tag{1}$$

To calculate the *Group Betweenness Centralization* index for a whole network, we need to compute the betweenness centrality probability index for each actor of the network. The probability index assumes that a "communica-

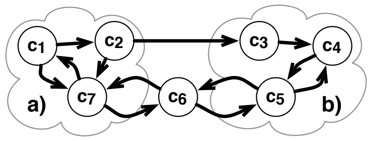

Figure 3. Example of a directed network to illustrate our social analysis measures.

tion" takes the shortest path from a contributor c_j to contributor c_k and if the network has more shortest paths, all of them have the same probability to be chosen. If g_{jk} is the number of shortest paths linking two contributors, $1/g_{jk}$ is the probability of using one of the shortest paths for communication. Let $g_{jk}(c_i)$ be the number of shortest paths linking two contributors that contain the contributors c_i. Freeman [11] estimates the probability that contributor c_i is between c_j and c_k by $g_{jk}(c_1)/g_{jk}$. The betweenness index for c_i is the sum of all probabilities over all pairs of actors excluding the ith contributor. Formula (2) shows the normalized betweenness index for directed networks.

$$C_B(c_i) = \frac{\sum_{j<k} g_{jk}(c_i)/g_{jk}}{(g-1)(g-2)} \qquad (2)$$

To compute a betweenness index for the complete network instead of a single node, we used Freeman's formula for *Group Betweenness Centralization*. The formula is shown in equation (3), in which $C_B(c^*)$ is the largest betweenness index of all actors in the network.

$$C_B = \frac{\sum_{i=1}^{g}[C_B(c^*) - C_B(c_i)]}{(g-1)} \qquad (3)$$

4.5.3 Structural holes

We use the following structural hole measures:

- The *Effective Size* of a node c_i is the number of its neighbours minus the average degree of those in c_i's ego network, not counting their connections to c_i. The effective size of node c_1 in Figure 3a is $2-1 = 1$. Note, that only direct neighbours of c_1 are considered and the directed connections are replace with undirected. The effective size of node c_4 in Figure 3b is $2 - 0 = 2$.

- The *Efficiency* normalizes the effective size of a node c_i by dividing the it's effective size with the number of it's neighbours. The efficiency of node c_1 in Figure 3a is $(2 - 1)/2 = 0.5$. The efficiency of node c_4 in Figure 3b is $(2 - 0)/2 = 1$.

- *Constraint* is a summary measure that relates the connections of a node c_i to the connections of c_i's neighbours. If c_i's neighbours and potential communication partners all have one another as potential communication partners, c_i is highly constrained. If c_i's neighbours do not have other alternatives in the neighborhood, they cannot constrain c_i's behavior.

To calculate network measures of the introduced ego network measures on structural holes, we compute the sum of the measures for each node of a network. As the measures are based on network connections, we normalize the sum by computing the fraction of the sum and the number of possible network connections.

4.6 Data collection

We mined the Jazz development repository for build and communication information. A query plug-in was implemented to extract all development and communication artifacts involved in each build from the Jazz server. These build-related artifacts included build results, teams, change sets, work items, contributors, and comments. We imported the resulting data into a relational database management system to handle the data more efficiently.

We extracted a total of 1288 build results, 13020 change sets, 25713 work item and 71019 comments. Out of a total of 47 Jazz teams, 24 had integration builds. The build results we extracted were created during the time range from November 5, 2007 to February 26, 2008.

Next, we had to make a decision for which builds and associated communication to analyze. Our selection criteria was that we analyze a number of build results that is large enough for statistical tests and include both `OK` and `ERROR` builds. Some teams used the building process for testing puroses only and created just a view build results, while others had either only `OK` or only `ERROR` build results. Predicting build results for a team that only produced `ERROR` builds in the past, will most likely yield an `ERROR`, since no communication information representing successful builds is available. Thus, we considered teams that had more than 30 build results and at least 10 failed and 10 successful builds. Five teams satisfied these constraints and were considered in our analysis. In addition, we included the nightly, weekly, and one beta integration build, although they did not satisfy our constraints, because they integrate all subsystems of the entire project.

5 Analysis and Results

Table 1 shows descriptive statistics of the considered builds and related communication networks of the five teams (B, C, F, P and W in the first 5 columns) and the nightly, weekly, and beta project-level integrations. For example, team B created 60 builds from which 20 turned out to be `ERROR`s and 40 `OK`. The communication networks of

	Team Level Builds					Project Level Builds		
	B	C	F	P	W	nightly	weekly	beta
# Builds	60	48	55	59	55	15	15	16
# ERRORs	20	16	24	29	31	9	11	13
# OKs	40	32	31	30	24	6	4	3
# Contributors:								
Min	3	9	6	5	13	43	37	55
Median	6	16.5	18	15	20	55	57	69.5
Mean	12.68	18.02	20.15	17.98	22.87	57.93	52.27	67.81
Max	58	31	64	61	52	75	75	79
# Directed Connections:								
Min	0	1	2	0	11	81	56	144
Median	13	39.5	95	36	74	236	149	280
Mean	51.58	53.4	87.78	63	88.35	253.1	171.9	285.8
Max	361	139	355	401	300	434	496	446
# Change Sets:								
Min	1	15	8	32	83	80	62	82
Median	10	38	35	46	111	117	115	178.5
Mean	10.83	44.38	42.65	47.25	115.3	129	114.2	166.8
Max	33	101	91	75	156	199	173	196
# Work Items:								
Min	0	2	1	1	10	11	5	31
Median	6.5	12	20	12	18	67	51	98
Mean	16.43	15.56	23.07	19.34	29.49	72.13	56.87	96.81
Max	131	50	100	107	119	132	202	170

Table 1. Descriptive build statistics.

this team had between 3 and 58 contributors (51.58 directed connections in average) and spanned 0 to 131 work items. The builds involved in average 10.83 change sets.

5.1 Individual communication measures and build results

To examine whether any individual measure of communication structure can predict integration failure or success (our Research Question 1), we analyze the builds from each team and project-level integration in part in relation to the communication structure measures as follows: For each team we categorize the builds into two groups. One group contains the ERROR builds and the other the OK builds. For each build and associated communication network we compute the network measures described in Section 4 and compare them across the two groups of builds (ERROR and OK).

The communication measures used in the analysis were: Density, Centrality (in-degree, out-degree, inout-degree, and betweenness), Structural Holes (efficiency, effective size, and constraint), and number of directed connections. We used the Mann-Whitney test [39] to test if any of the measures differentiate between the groups of ERROR and OK related communication networks. We used the α-level of .05 and applied the Bonferroni correction to mitigate the threat of multiple hypothesis testing. None of the tests yielded statistical significance, which indicates that no individual communication structure measures significantly differentiate between ERROR and OK builds.

We also tested for the possible effect of the technical measures shown in Table 1: #Contributors, #Change Sets and the #Work Items on the build result. Also, none of the tests yielded statistical significance to differentiate between ERROR and OK builds.

		prediction	
		OK	ERROR
actual	OK	26	5
	ERROR	9	15

Table 2. Classification results for team F.

5.2 Predictive power of combined measures of communication structures

The second research question aims to assess whether the combination of communication related network measures can predict future build results. Thus we combined communication structure measures analyzed in RQ1 into a predictive model that classifies a team's communication structure as leading to an ERROR or OK build. We explicitly exclude the technical descriptive measures such as #Contributors, #Change Sets and the #Work Items from the model in order to focus on the effect of communication on build failure prediction. We validate the model for each set of team-level and project-level networks separately by training a Bayesian classifier [20] and using the *leave one out cross validation* method [20].

For example, to predict the build result N of team F's 55 build results, we train a Bayesian classifier with all other 54 build results and their communication related network measures. Then, we input the communication measures of Build N's related communication network into the classifier and predict the result of build N. We repeat the classification for all 55 builds of team F and sum up the number of correctly and wrongly classified results.

Table 2 shows the classification result for team F. The upper left cell represents the number of correctly classified communication networks as related to OK builds (26 vs. 31 actual), and the lower right cell shows the number of correctly classified networks as leading to ERROR builds (15 vs. 24 actual). The other two cells show the number of wrongly classified communication networks.

The classification quality is assessed via recall and precision coefficients, which can be calculated for ERROR and OK build predictions. We explain the coefficients for prediction of ERROR builds.

Recall is the percentage of correctly classified networks as leading to ERROR divided by the number of ERROR related networks. In Table 2 the lower right cell shows

	Team Level Builds					Project Level Builds		
	B	C	F	P	W	nightly	weekly	beta
ERROR Recall	.55	.75	.62	.66	.74	.89	1	.92
ERROR Precision	.52	.50	.75	.76	.66	.73	.92	.92
OK Recall	.75	.62	.84	.80	.50	.50	.75	.67
OK Precision	.77	.83	.74	.71	.60	.75	1	.67

Table 3. Recall and precision for ERROR and OK build results using the Bayesian classifier.

the number of correct classified networks that are leading to ERRORs, which is divided by the sum of the values in the lower row, which represents the total number of actual ERRORs. This yields for Table 2 a recall of $15/(9+15) = .62$. In other words, 62% of the actual to ERROR leading networks are correctly classified.

Precision is the percentage of as to ERROR leading classified networks that turned out to be actually ERRORs. In Table 2, it is the number of correctly classified ERRORs divided by the sum of the right column, which represents the number of as ERROR classified builds. In Table 2 the precision is $15/(5+15) = .75$. In practical terms, 75% of the ERROR predictions are actual ERRORs.

We repeated the classification described above for each team and project-level integration. Note that the model prediction results only show how the models perform within a team and not across teams. Table 3 shows the recall and precision values for as to OK and ERROR leading classified communication networks for each of the five team-level and three project-level integrations. Since we are interested in the power of build failure prediction, the error related values from our model are of greater importance to us. The ERROR recall values (how many ERRORs were classified correctly) of team-level builds are between 55% and 75% and the recall values of the project-level builds are even higher with at least 89%. The ERROR precision values are equally high.

6 Threats to validity

We conceptualized *communication* based on comments on work items. Besides that, the Jazz team communicates via email, chat, web-based information and face-to-face meetings. Based on our observations and conversations with the Jazz team, we are certain that comments are mostly used to communicate about work items. Since they are work item-specific and immediately available.

For the *network construction*, we assumed that every developer commenting on or subscribed to a work item reads all comments of that work item. This assumption might not always be correct. By manual inspection of a selected number of work items, we found that developers who commented on a work item are aware of the other comments, confirming our assumption.

Due to storage problems the Jazz teams erased some build results. In the case of nightly builds we expected 90 builds (according to project duration) but found only 15. This might affect our results but we argue that due to our richness of data the general trend is still preserved. Hence we expect that our findings would be supported by the complete data, too.

Our results are only valid for the Jazz project and can not be generalized to any software development project. Additional studies on different projects using Jazz could be conducted to confirm our results.

7 Discussion

As far as we know, our study is one of the first to investigate communication structures at the level of integration builds. This is a finer-grain level of analysis than that of current studies in the literature, which found that coordination failures are due to communication problems. By conceptualizing coordination failure as a failed integration result, we studied the role of communication structures in successful coordination. In Section 7.1 we discuss our contribution which is two-fold: (1) while communication structure measures can not individually predict failed builds, (2) a set of communication structure measures can be combined into a model that predicts failed integration builds. Subsequently, we discuss practical implications of our work in Subsection 7.2.

7.1 Communication as predictor for integration failure

In our analysis we examined the relationship between integration builds and measures of the related communication structure. We found that none of the single communication structure measures (density, centrality or structure hole measures) significantly differentiated between failed and successful builds at the team-level and project-level. Therefore none of these individual communication structure measures could be used to predict integration build results.

In addition to the communication related measures, we also examined whether the technical measures we computed when constructing the communication networks – the number of change sets, contributors, and work items – have an impact on the integration build result, as they are an indication for the size and complexity of the development tasks to be coordinated. According to Nagappan and Ball [31], one might expect that increased size and complexity of

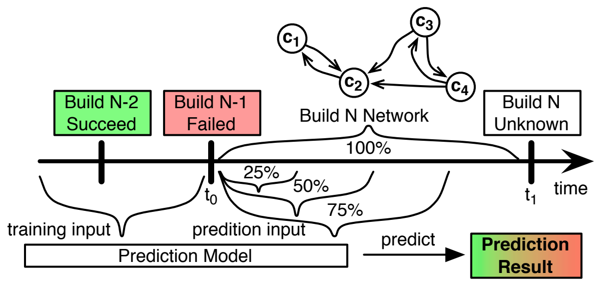

Figure 4. Predicting Build N Result with different reduced communication data.

code changes relates to more build failures. But in our study these single measures did not significantly differentiate between successful and failed build results. However, additional technical measures that were used by Nagappan might be good predictors in Jazz as well.

The second contribution of this work is the predictive model that uses measures of communication structures to predict build results. Interestingly, the combination of communication structure measures was a good predictor of failure even when the single measurements were not. Our model's precision in predicting failed builds, which relates to the confidence one can have in the predicted result, ranges from 50% to 76% for any of the five team-level integration builds, and is above 73% for the project-level integration builds.

We found that, for all prediction models, the recall and precision values are better than guessing. A guess is deciding on the probability of an `ERROR` or an `OK` build if the build fails or succeeds. The probability is the number of `ERROR`s or `OK`s divided by the number of all builds. For example, if we know that the `ERROR` probability is 50% and we guess the result of the next build we would achieve a recall and precision of 50%. In our case, our model reached an `ERROR` recall of 62% for team F, where as a guess would have yield only $24/55 = .44 = 44\%$ (see Table 1).

7.2 Practical implications

Our model can be used by Jazz teams to assess the quality of their current communication in relation to the result of their upcoming integration. If a team is currently working on a component and an integration build is planned in the near future, the measures of the current communication in the team can be provided as input to our prediction model and the model will predict whether the build will fail with a precision shown in Table 3. For example, if team P is working towards a build and our model predicts that the structure of its current communication leads to a failed build, the team can have a 76% (see Table 3) confidence that the build is going to fail. This information can be used by developers in monitoring their team communication behavior, or by management in decisions with respect to adjusting collaborative tools or processes towards improving the integration.

In our analysis we used the complete communication of the build related time interval as input for the model to predict the build result. In Figure 4, this is the time interval from t_0 to t_1 for $Build\ N$. This time interval might not be appropriate for practical applications in which the build process takes only a view minutes, as t_1 is immediately before the build starts. Predicting the build result has no value for the developers when they have no time left to react on a build failure prediction and they can simply run the build instead of predicting the result. For long builds that last several days, the prediction at t_1 is still practical relevant. Developers might delay the build to perform reactive changes on the failure prediction.

In an additional analysis we investigated the prediction recall and precision by only considering the communication of reduced time intervals as input for our model. For each build N result, we trained our model with the complete communication of all build results except the one from N, and predicted the result N by only considering the first 25%, 50% and 75% of the communication as input for the prediction model (see Figure 4).

We only show the precision and recall of the prediction for the 25% time range in Table 4, as they are already high enough for practical use. We observe that the model has almost the same predictive power as the model considering the complete time range of the build. A possible explanation is the involvement of work items in many builds. That means that the task completion time of a work item is longer than the time interval of a build and task related change sets and communication occurs in multiple builds. When a developer delivers a change set in the first 25% of the build related time interval and the change set is associated to a work item that has already been discussed, the complete past communication is included. Including the past communication is valid, as our prediction model learns from the

	Team Level Builds					Project Level Builds		
	B	C	F	P	W	nightly	weekly	beta
`ERROR` Recall	.63	.69	.62	.62	.68	1	.45	.85
`ERROR` Precision	.43	.46	.68	.75	.64	.75	.83	.92
`OK` Recall	.60	.59	.77	.80	.50	.50	.75	.67
`OK` Precision	.77	.79	.73	.69	.55	1	.33	.50

Table 4. Recall and precision results only including first 25% of the communication.

communication history. In Jazz, a work item is in average included in 5.17 builds (min=1, median=3, max=57), which provides a possible explanation for having valuable prediction results even early in the build related time interval.

With the results of this analysis, we are confident that our prediction model is practically useful, even if it does not consider the complete communication of a build. Thus, developers and managers have time to react on build failure predictions before the actual build fails.

8 Conclusion

Our results indicate that developer communication plays an important role in the quality of software integrations. Complementary to existing literature on communication and coordination outcome in software engineering, we objectively measured coordination outcome by the result of the integration build. To capture and study communication we used social network measures. Although we found that no individual social network measure could indicate whether a build will fail or succeed, the combination of network measures can be combined into a predictive model. The model indicates whether an integration will fail based on the current communication structure of a development team. For five project studied teams, our predictive model yielded recall values between 55% and 75%, and precision values between 50% to 76%.

The study of communication patterns in relation to team performance is important given that communication is the main mechanism for knowledge sharing in work groups. Software engineers differ in the expertise, knowledge, and background they bring to the development task. Group performance depends not only on the information available to developers and the knowledge distribution within the team, but also on the properties of the communication structure that facilitates knowledge dissemination.

Having identified that communication plays an important role in integration success, future research should examine the interaction with other factors that could determine integration failure. In addition to our focus of communication, the analysis of code and code changes should be considered. We believe that a combination of both will lead to even better prediction models for software integration builds.

Based on the information provided by our predictive model, awareness systems can be developed to provide recommendations to improving a team's communication structure when developers face a likly build failure. We intend to integrate our predictive model into the Jazz development environment itself, and provide developers and managers with a real-time communication quality feedback mechanism that would predict possible failed builds based on the current communication behavior in the project. This will give us the opportunity to evaluate the value of our predictions with respect to improvements in software coordination.

9 Acknowledgements

We thank all members of the IBM Jazz project and SEGAL Group. Funding for this project was provided by NSERC and IBM. IBM and Jazz are trademarks of IBM Corporation in the US, other countries, or both.

References

[1] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Softw. Eng.*, 22(10):751–761, 1996.

[2] R. M. Bell. Predicting the location and number of faults in large software systems. *IEEE Trans. Softw. Eng.*, 31(4):340–355, 2005.

[3] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In *Proc. of the 2006 Int. workshop on Mining Soft. repositories*, pages 137–143, 2006.

[4] R. Burt. *Structural Holes: The Social Structure of Competition*. Harvard University Press, August 1995.

[5] M. Cataldo, M. Bass, J. D. Herbsleb, and L. Bass. On Coordination Mechanisms in Global Software Development. In *Proc. of the 2nd Int. Conf. on Global Soft. Eng.*, pages 71–80, 2007.

[6] B. Curtis, H. Krasner, and N. Iscoe. A Field Study of the Software Design Process for Large Systems. *Communications of the ACM*, 31(11), 1988.

[7] D. Damian, L. Izquierdo, J. Singer, and I. Kwan. Awareness in the wild: Why communication breakdowns occur. In *Proc. of the 2nd Int. Conf. on Global Soft. Eng.*, pages 81–90, 2007.

[8] C. R. B. de Souza, D. Redmiles, L.-T. Cheng, D. Millen, and J. Patterson. How a good software practice thwarts collaboration: the multiple roles of apis in software development. In *Proc. of the 12th ACM SIGSOFT Int. symposium on Foundations of Soft. Eng.*, pages 221–230, 2004.

[9] C. R. B. de Souza, D. Redmiles, L.-T. Cheng, D. Millen, and J. Patterson. Sometimes you need to see through walls: a field study of application programming interfaces. In *Proc. of the Computer Supported Cooperative Work*, pages 63–71, 2004.

[10] G. Denaro, S. Morasca, and M. Pezzè. Deriving models of software fault-proneness. In *Proc. of the 14th Int. Conf. on Soft. Eng. and knowledge Eng.*, pages 361–368, 2002.

[11] L. C. Freeman. Centrality in social networks: Conceptual clarification. *Social Networks*, 1(3):215–239, 1979.

[12] R. Frost. Jazz and the Eclipse Way of Collaboration. *IEEE Software*, 24(06):114–117, 2007.

[13] S. R. Fussell, R. E. Kraut, F. J. Lerch, W. L. Scherlis, M. M. McNally, and J. J. Cadiz. Coordination, overload and team performance: effects of team communication strategies. In *Proc. of the Computer Supported Cooperative Work*, 1998.

[14] P. A. Gloor, R. Laubacher, S. B. C. Dynes, and Y. Zhao. Visualization of communication patterns in collaborative innovation networks - analysis of some w3c working groups. In *Proc. of the 12th Int. Conf. on Information and knowledge management*, 2003.

[15] C. N. Gonzalez-Brambila and F. Veloso. Social capital in academic engineers. In *Portland Int. Center for Management of Eng. and Technology*, 5-9 Aug. 2007.

[16] A. Griffin and J. R. Hauser. Patterns of Communication Among Marketing, Engineering and Manufactoring–A Comparison Between Two New Product Teams. *Management Science*, 38(3), 1992.

[17] R. E. Grinter, J. D. Herbsleb, and D. E. Perry. The Geography of Coordination: Dealing with Distance in R&D Work. In *Proc. of the Int. Conf. on Supporting Group Work*, pages 306–315, 1999.

[18] A. Hargadon and R. I. Sutton. Technology brokering and innovation in a product development firm. *Administrative Science Quarterly*, 42(4), 1997.

[19] A. E. Hassan and K. Zhang. Using Decision Trees to Predict the Certification Result of a Build. In *Proc. of the 21st IEEE/ACM Int. Conf. on Automated Soft. Eng.*, pages 189–198, 2006.

[20] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 3 edition, July 2003.

[21] J. D. Herbsleb and R. E. Grinter. Splitting the organization and integrating the code: Conway's law revisited. In *Proc. of the 21st Int. Conf. on Soft. Eng.*, pages 85–95, 1999.

[22] J. D. Herbsleb and A. Mockus. An Empirical Study of Speed and Communication in Globally Distributed Software Development. *IEEE Transactions on Soft. Eng.*, 29(6):481–494, June 2003.

[23] J. D. Herbsleb, A. Mockus, and J. A. Roberts. Collaboration in Software Engineering Projects: A Theory of Coordination. In *Int. Conf. on Information Systems*, 2006.

[24] P. Hinds and C. McGrath. Structures that Work: Social Structure, Work Structure and Coordination Ease in Geographically Distributed Teams. In *Proc. of the 20th Conf. on Computer Supported Cooperative Work*, pages 343–352, 2006.

[25] H. Holmstrom, E. O. Conchuir, P. J. Ågerfalk, and B. Fitzgerald. Global Software Development Challenges: A Case Study on Temporal, Geographical and Socio-Cultural Distance. In *Proc. of the 1st Int. Conf. on Global Soft. Eng.*, pages 3–11, 2006.

[26] L. Hossain, A. Wu, and K. K. S. Chung. Actor centrality correlates to project based coordination. In *Proc. of the 20th anniversary Conf. on Computer Supported Cooperative Work*, pages 363–372, 2006.

[27] S. Kim, J. Whitehead, and Y. Zhang. Classifying software changes: Clean or buggy? *IEEE Transactions on Soft. Eng.*, 34(2):181–196, 2008.

[28] R. E. Kraut and L. A. Streeter. Coordination in Software Development. *Communications of the ACM*, 38(3):69–81, March 1995.

[29] A. Meneely, L. Williams, W. Snipes, and J. Osborne. Predicting Failures with Developer Networks and Social Network Analysis. In *Proc. of the 16th ACM SIGSOFT Int. Symposium on Foundations of Soft. Eng.*, 2008.

[30] A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, 2000.

[31] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In *ICSE '05: Proc. of the 27th Int. Conf. on Soft. Eng.*, May 2005.

[32] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proc. of the 28th Int. Conf. on Soft. Eng.*, pages 452–461, 2006.

[33] M. B. Pinto and J. K. Pinto. Project Team Communication and Cross-Functional Cooperation in New Program Development. *Journal of Product Innovation Management*, 7(3), 1990.

[34] M. Pinziger, N. Nagappan, and B. Murphy. Can Developer Social Networks Predict Failures? In *Proc. of the 16th ACM SIGSOFT Int. Symposium on Foundations of Soft. Eng.*, 2008.

[35] R. Reagans and E. W. Zuckerman. Networks, Diversity, and Productivity: The Social Capital of Corporate R&D Teams. *Organization Science*, 12(4), 2001.

[36] D. L. Rulke and J. Galaskiewicz. Distribution of Knowledge, Group Network Structure, and Group Performance. *Management Science*, 46(5):612–625, 2000.

[37] A. Sarma and A. van der Hoek. Towards awareness in the large. In *Proc. of the Int. Conf. on Global Soft. Eng.*, pages 127–131, 2006.

[38] A. Schröter, T. Zimmermann, and A. Zeller. Predicting Component Failures at Design Time. In *Proc. of the 5th Int. Symposium on Empirical Soft. Eng.*, pages 18–27, September 2006.

[39] S. Siegel. *Nonparametric Statistics for the Behavioral Sciences*. 1st edition, 1956.

[40] W. Souder. Managing Relations Between R&D and Marketing in New Product Development Projects. *Journal of Product Innovation Management*, 5(1), 1988.

[41] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, November 1994.

[42] T. Zimmermann and N. Nagappan. Predicting defects using network analysis on dependency graphs. In *Proc. of the 30th Int. Conf. on Soft. Eng.*, pages 531–540, May 2008.