Awareness in the Wild: Why Communication Breakdowns Occur

Daniela Damian, Luis Izquierdo, Janice Singer* and Irwin Kwan
Dept of Computer Science, University of Victoria
PO Box 3055, Victoria, BC V8W 3P6, Canada
{ danielad, luis, irwink@}@cs.uvic.ca

**National Research Council Canada*
Montreal Rd., Bldg M50, Ottawa, ON K1A 0R6
janice.singer@nrc-cnrc.gc.ca

Abstract

Global software teams face challenges when collaborating over long distances, such as communicating changes in the project. During a four-month case study at IBM Ottawa Software Lab we observed the collaboration patterns of a multi-site development project team. In this period, we inspected project documentation, interviewed team leaders, attended project meetings, and spoke with developers to identify problems originated by the lack of awareness of changes related to the implementation of work items. Our observations show (1) that organizational culture has an effect on how developers are made aware; (2) that communication-based social networks revolving around particular work items are dynamic throughout development, and therefore awareness needs to be maintained in infrastructures of work; and (3) that information overload and communication breakdowns contributed to the generation of a broken integration build. We discuss these breakdowns in communication and implications for the design of collaboration tools that could mitigate these problems.

1. Introduction

Awareness is becoming an important topic of research in software engineering. Frequently defined as "an understanding of the activities of others, which provides a context for one's own activities" [6], in software engineering, awareness is generally linked to issues related to coordination, or managing dependencies between activities [15]. Awareness is seen as a means by which team members can become aware of the work of others that is interdependent with their current tasks, therefore enabling better coordination of teams. Examples of task awareness include knowing who has most recently worked on a particular piece of code, or knowing who has what code checked out of a repository at a particular time, or knowing who is expert on a particular module or section of code.

Several recent systems have been developed ([16][4][10][18][2]) that attempt to increase the awareness of individuals as they work in teams. These systems focus on identifying various pieces of information (such as where conflicts may occur), that are thought to be of value to software engineers, and then using awareness mechanisms to disseminate/convey information to interested or relevant individuals.

Most of these systems are focused on small or collocated teams. As Sarma and van der Hoek [17] point out, global software development (GSD) teams have special needs with respect to "awareness in the large." Geographically distributed software development is becoming a business necessity and common practice in most large companies. Yet, it is unclear if the awareness systems proposed for small teams will support development in GSD. Furthermore, while the awareness needs of GSD teams appear to be greater than those of small co-located teams, an understanding of those needs 'in the wild' has not yet been explored. Before we can build useful tools and processes, we need to gain insights into the awareness needs of GSD teams and their role in supporting more effective coordination.

Our goal in this paper is thus to investigate and characterize awareness in GSD, especially at the task level**.** To do so, we conducted a case study of a globally distributed team of developers at IBM. Specifically we were interested in studying the consequences of gaps in awareness and in examining possible relationships between broken builds and problems in awareness.

2. Related work

Much of the work in the area has focused on communication and coordination issues in software teams. Curtis et al. [5] studied how communication networks and breakdowns affected software development. Their findings raised many issues that are critical for awareness systems, such as the importance of both formal and informal communication in development. Gutwin, et al. [10] looked at several open source development projects and found that developers needed to maintain awareness of others to contribute to development, and that they did so primarily through text based communication. Herbsleb and Grinter [12] conducted a field study showing the importance of informal communication, and furthermore the difficulty in communicating across globally distributed teams– suggesting that an increase in awareness would benefit development. Others have looked at the mismatch between coordination requirements and actual communication [3][7] and proposed mechanisms to improve the mismatches that occur. Espinosa, et al [8] have identified factors that affect awareness in software development, including the nature of team knowledge and distance.

International Conference on Global Software Engineering(ICGSE 2007)
0-7695-2920-8/07 $25.00 © 2007 IEEE

Although these studies have pointed out many potential requirements for awareness systems, they unfortunately do not specifically investigate specific collaboration and coordination situations and how they are affected by awareness. Therefore, we used an approach not widely used in software engineering to examine collaboration by charting social networks to understand who communicated with whom, what information was transferred, and so on. Another difference from existing literature is our in-depth "in the wild" examination of a large project, rather than reliance on interviews, document inspection, or project history.

We report three critical findings related to building awareness in GSD. First, organizational culture differs across distributed teams involved in the same project and has an impact on how awareness is managed. Second, social networks revolving around particular work items are dynamic and evolve depending on development cycle, thus awareness needs to be maintained within organic infrastructures of work. Finally, inadequate communication and coordination mechanisms can themselves lead to problems in awareness – illustrated in the study by an example of communication overload leading to a broken build. These findings have implications for the design of awareness mechanisms as we discuss in our conclusions.

Our paper is structured as follows. First, we provide an overview of our research process and data collection methods. Second, we provide a detailed account of each of our findings. Next, we interpret the findings with respect to requirements for awareness in GSD, and how they may differ from collocated teams. Finally, we propose a characterization for awareness needs in GSD.

3. Research Design

We conducted a case study of a distributed software development team at the IBM Ottawa Software Lab. We conducted on-site observations and interviews with the development team and managers during a 4-month period, September-December 2005.

Our research approach was to study the collaboration practices of the distributed team in order to understand how distributed developers maintain awareness, what lack of awareness means, and what the consequences are of lack of awareness. Specifically, in studying the collaboration of distributed teams working on system features we wanted to gain more understanding of the membership of collaboration networks around features (i.e. who are the contributors to feature development, what information they exchange for cross-site coordination), whether this communication of awareness information is timely, and whether there are any gaps that lead to ineffective coordination.

3.1. Study setting: the company, the project and its process, and the distributed team

At IBM, we had access to a large distributed development project that required extensive cross-site communication and coordination of technical work.

The project had fifty developers in a total of nine sites in the USA, Europe and Canada. To study aspects of awareness and in particular of change information propagation between developers working on the same feature, we focused the investigation on the implementation of a major feature and the collaboration patterns of the members associated with the feature. This involved two of the nine locations, one in Canada and one in United States. The site located in Canada is the one leading the development, and the site in US recently joined the project. These two sites are involved in the implementation of one of the most important components in the system (also referred to as a feature). We call this component the CSM component for confidentiality reasons. The development of this component required decision-making and cross-site synchronization on a daily basis. The Canada and US teams had not previously worked together and each team had a different organizational culture, even though both teams were a part of IBM. The Canadian team had extensive experience in distributed software development, as part of the Eclipse IDE team - in the past, they have continuously coordinated with several IBM sites in the US and Europe. On the other hand, the American team was new to distributed software development, since the product-line that they had previously worked on did not require much interaction with remote teams.

3.1.1. Development Process

In the project studied, a project plan is divided into features, of which development is further subdivided into work items. A work item is assigned to one or more developers for implementation.

The work item implementation process typically involves a design stage during which a number of developers are assigned to create and contribute to an *implementation proposal*, and a coding stage during which this proposal is implemented. When an implementation proposal is drafted, it is posted to a Bugzilla tracking system for review by other developers. One of the assigned developers posts a comment in Bugzilla to alert team members about the availability of the implementation proposal, which is then accessed for review and comments during a stage referred to as the *design phase*. Once the proposal is reviewed, the assigned developers implement the work item and check it in when it is completed.

3.1.2. Build Generation Process

To investigate the consequences of awareness with respect to coordination problems, we investigated each broken build that occurred during our research. A broken build, in addition to being an easily-observable event, has a large negative effect on development. Broken builds lead to lost productivity because of the need to roll back changes, and the delays caused to developers' work. We focused on broken builds because of a lack of other metrics, such as hours of rework. Broken builds are also indicative of errors in development and lower quality in the software product.

The project generated two types of builds for the software being developed: continuous and integration builds Generated every 30 minutes, continuous builds are done on per-component basis, so that each component has continuous feedback and catches build breakers quickly; red flags in continuous builds are not as critical as in integration builds. Weekly integration builds, on the other hand, compile all the components of the whole project. Keeping integration builds error-free is one of the concerns of the whole team.

3.2. Data sources, collection and analysis methods

During a period of four months one researcher was located at the IBM Ottawa Software Labs, and observed the daily interaction between the local team members involved in the CSM component development. Additionally, the researcher was allowed to participate in most teleconferencing project meetings with the distributed team, and was also included in the project mailing list discussions. Specifically, the data sources and data collection and analysis methods we used are as follows:

Project Documentation Review. Most of the project documentation generated in the project was posted on a WIKI that was accessible to the researcher. The project plan was used to identify the developers assigned to each of the work items. The design documentation was inspected on a daily basis to track changes to the design and identify the contributing developers.

Interviews. Interviews with the local Ottawa based project members were held during the four-month period, almost on a daily basis. In the beginning of the study, our goal was to identify a software component whose development required high interaction among distributed team members and to understand which of our data collection methods were allowed by the project team. A first interview with the project manager was thus conducted to understand project characteristics. Subsequently, the CSM component and its team were identified as the target for our study. We then interviewed the CSM team leader in order to obtain a picture of the component development complexity, and to define the best approach for data collection. Ongoing interviews, mostly unstructured and informal, were then conducted throughout the four-month stay with the team.

Monitoring of discussions in the bug tracking system. Bugzilla was used in this project not only as a bug tracking system, but also to host discussion about design and implementation of work items. Work items were assigned to developers by creating a bug ticket in Bugzilla. We monitored the discussions on all work items to identify contributors associated with the design and implementation of each work item. Bugzilla was one of the two main data sources used to determine each work item's normative plan as well as the communication-based social network at design stage (see Figure 1).

Activities log compilation. Project members were invited to keep a diary of their interactions with other team members. An interaction log was designed to record, for each interaction: the work item number, name of the contact person, reason for the communication, and the media used to reach the person. See Table 1 for a sample of the data requested from a developer. The names have been changed to preserve anonymity. The interaction logs were the other main source of information for constructing the social network shown in Figure 2.

Six developers agreed to provide this interaction data, four in Ottawa and two at the US site. To collect this information, the on-site researcher met every day with each of the four developers located in Ottawa where they reported their daily interactions. The two US developers sent their logs by email each day for a period of five business days. While the process of interaction recording relied on retrospection for the Ottawa developers, the US developers agreed to record instances of interaction when they occurred.

Table 1: Excerpt from the interaction log provided by Brian for Nov 9th

Name: Brian			**Nov 9th**
Location: Ottawa			
Work Item	**Contact person**	**Reason of communication**	**Media Used**
Bug 0001	Trish	Update of the module 1 development	email
	Fabio	Coordination of the implementation	phone
	Chris	API issues and semantic checks	Face to face
Bug 0002	Florian	Discussion and comments about the new proposal	chat

4. Findings

Three findings stood out in our analysis of the data. First, as discussed in Section 4.1, organizational culture played a role in the effective coordination of development and maintenance of awareness of work across development sites. Second, as discussed in Section 4.2, we found that the development of features involves a dynamic and evolving communication-based social network that is different than the one indicated in the task allocation plan.

Finally, we present in Section 4.3 an investigation of collaboration patterns in the development of a work item that indicates a relationship between awareness mechanisms within dynamic social networks and coordination problems. Specifically, we see that an overload of information caused a broken integration build.

4.1. Lack of awareness due to differences in organizational cultures

In large companies such as IBM, there is a diversity of products, technologies, teams and processes, and consequently organizational cultures—affecting both working and communication styles. In our case study, we observed how social factors, such as organizational culture and history of the relationship between sites, play an important role in awareness of changes during the development of work items that require cooperation and coordination

across sites. As an example, we highlight the differences in organizational culture between the U.S. and the Canadian team in terms of the process followed to communicate changes in a new build.

We identified this by analyzing email threads in the mailing list. We looked for messages related to a communication problem, then followed the subsequent discussion, and evaluated its relevance to the task awareness context.

The two teams involved in the development of the CSM component had not worked together before, and the process of the US team joining the project required a long negotiation. Previously, the team in US was involved in the development of a similar feature; hence the members of the US team felt they could and wanted to play an important role in the overall development of this project.

Email messages posted on CSM's mailing list indicate that changes to artifacts were not communicated due to false assumptions the Canadian team made about the US team. More precisely, developers at the Canadian site assumed that their remote peers always refer to the build notes to review the list of changes, while, in fact, the US site relied on email lists to communicate changes regardless of their importance.

To illustrate these observations, the following messages are included from a thread posted in the CSM mailing list; the discussion provides some insights about a particular case where developers from both teams found a problem caused by the lack of awareness of changes.

One of the US developers found that his work was affected due to changes made by a Canadian developer, and he was not alerted in a timely fashion of this change:

> Developer (US) : "When I came in this morning, expecting to be able to deliver after last evening's work, I was thwarted by changes that I wasn't expecting. That pushed back my schedule, in turn pushing back [other developer]'s schedule."

Moreover, the concerns of this developer go further when he notes that there has not been any discussion about the process of communication of changes in the code, i.e. the introduction of new classes:

> Developer (US): "The second issue is the introduction of new classes that I didn't hear any discussion on. Maybe I missed it. Or maybe the classes were deemed too trivial to comment on - that's fine. However, in **the culture I work** in, folks send out mail on new classes… Sure, I can see it in the sync report, but I think it helps to get a head's up on it, esp. when we're not sitting in the same office" (bold is our emphasis).

Of note is the explicit mention of the existence of a culture perceived as being different than that found at other development sites. The US developer indicates that, due to the team distribution, they cannot rely only on the code synchronization reports; instead they would prefer to use a mechanism that alerts them about the changes, regardless if a log of changes is generated during the build creation.

This implicit proposition of using email as a primary communication tool to propagate changes at different levels of granularity generates a reaction at the Canadian site:

> Developer 1 (Ottawa): "**To me, emailing every intention/commit is overkill**. Is there not somewhere in between that we can meet?" (bold is our emphasis).

The reaction of the Ottawa developer shows concern about the volume of communication that could be handled by an individual. The team members at both sites finally agree that the use of email to communicate any minor change is overkill, yet acknowledge the need to somehow address communication style in lieu of their distributed relationship:

> Developer 2 (US): "I sympathize with your feeling that "check-in email" for every minor change is overkill. I also think that given the newness of our distributed team, we need to err on the side of over communication for the time being."

Although developers agree that the use of email could harm the productivity:

> Developer 2 (Ottawa): "I should spend more time coding than communicating",

they are aware that they have to reach a balance between amount of communication and other activities, and that technology plays an important role in how communication will be propagated among team members. Other solutions are proposed within the existing project infrastructure.

> Developer 2 (US): "Is there a way for the CVS server to be configured to send check-in email, such that if we put useful comments into the commit dialog box, others would see it? CVS has such a feature …"
>
> Developer 2 (Ottawa): "… Rather than explicit emails, I would be happier with updating the bugzilla entries, so to consolidate discussion and to avoid interaction with lotus notes. "

The time spent in discussing the role of email messaging for change notification finally pays off: team members in Ottawa and the US agreed to use Bugzilla and its notification functionality as a mechanism to keep members alerted about changes in the implementation of work items.

The cultural differences between the US and Ottawa teams highlight challenges for any awareness system. Although ostensibly both teams are part of the larger IBM culture, the specific organizational culture across the teams varies greatly, especially with respect to how to communicate changes. Thus, when considering the awareness needs of distributed teams, it is not enough to focus on just one site. Rather, each of the teams in the distributed partnership may have special needs or requirements with respect to awareness. The cross-team communication styles may need to be negotiated such that they are amenable across the cultural divide.

Furthermore, a major challenge in the design of effective awareness mechanisms is providing the appropriate

amount and quality of awareness information to the relevant developers without introducing extraneous information that affects their productivity. Interview data indicates that developers often hesitate using email because of the perception of leaving a copy of the communication in the email server of the company, which could be used to measure performance.

The lack of knowledge about how to communicate changes harms the trust relationship between teams, and slows down the integration process of a new team into development. Here, this lack of trust was exacerbated by differences in organizational culture and history of the relationships between the two sites. An interview with the CSM component team leader, indicates that the implementation of this component was particularly challenging, not only because of technical reasons, but also because the Ottawa team was the lead-team chosen by management, although there was a perception that the US team, according to its credentials, should have been chosen.

In summary, awareness needs are at least partly determined by organizational culture. Because different sites may have their own organizational cultures, one challenge is to build an awareness mechanism that can handle differences in organizational culture and process across distributed teams.

4.2. Dynamic, Emergent Teams in Software Development

To better understand how people coordinate in a software engineering workplace, we tracked the evolution of eleven work items in the project. The information collected in the interaction logs and the Bugzilla tracking system allowed us to construct communication-based social networks showing interaction patterns during the implementation of these work items. We were interested in gaining a deeper understanding of (1) the membership of collaboration networks around work items (i.e. who are the contributors to specific work items, what information they exchange for cross-site coordination), and (2) any gaps in collaboration and awareness that lead to coordination breakdowns.

To investigate the collaboration patterns within these social networks, we chose to focus on a work item (WI 1009) that was part of the core functionality of the CSM, which had high coordination needs with several other API clients, and for which we observed a broken integration build. Moreover, WI 1009 was chosen for analysis because its integration error at the weekly integration build was due to communication problems, whereas other broken builds were due to technical problems.

We tracked the evolution of the WI over its development cycle of 19 days, from its inception until its delivery to the CVS code repository to observe the collaboration of project members and examine aspects of awareness of work within the team. In particular, we wanted to study whether developers are aware of who else is contributing to the development of this work item, i.e. know who whose work affects their own, and whether they notify or

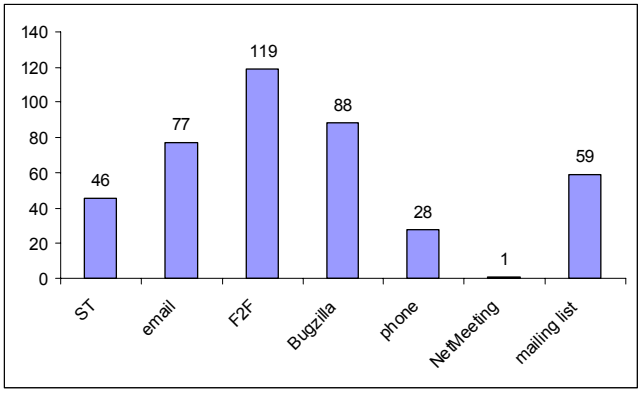

Chart 1. Number of times each communication medium was used, from interaction logs, Bugzilla, and mailing lists.

get notified about changes that affect their work and thus ultimately avoid broken builds. In this direction, we collected information on the project members that contributed to different phases of the development of WI 1009 (design, implementation and integration) and their communication patterns. Social networks visualizing instances of collaboration around WI 1009 at these distinct stages in its lifecycle were created and show interesting characteristics that aided our analysis of awareness within a dynamic team. This is discussed in Section 4.2.2.

Before that, however, Section 4.2.1 includes some details on the communication media and patterns of use for this particular work item, to provide context to better understand the insights described in Section 4.2.2.

4.2.1. Communication Patterns for WI 1009

Information on project communication behavior (communication tools used, who communicated with whom and for which reason, whether in the local or with remote colleagues) was gathered by means of the interaction log described in Section 2.2, over the period of 19 business days. The criterion for the data compilation was the same for local and remote teams.

If an interaction involving more than one developer was mentioned by multiple respondents, we counted it only once. A total of 418 unique interactions among developers were recorded for the 19-day time period, either through the interaction logs or from inspections of the mailing list and the Bugzilla system (one posting in Bugzilla or the mailing list was recorded as one interaction). Out of 418 interactions, 271 of the interactions were reported in the interaction logs; further, 26 out of the 271 interactions included more than one communication media. The communication media reported in the interaction logs were: SameTime, email, face to face (F2F), phone and NetMeeting. Chart 1 shows the number of times each of these media was reported as used in the interaction logs, together with the number of postings in Bugzilla and in the mailing list.

Table 2 shows the actual breakdown of the reported interactions as well as the Bugzilla postings by the six re-

spondents (Brian, Zoe, Chris, and Lucas, from the Canadian site, and Trish and Joshua from the US site). The names have been changed to preserve anonymity. The communication in Bugzilla (separate from the interaction logs) is included to illustrate an interesting pattern in using Bugzilla in this project. Typically a bug tracking and not communication system, Bugzilla was used to host discussion about the design proposals and implementation. As shown in Chart 1, the 88 postings counted in Bugzilla represent 21% of the total project communication. In the absence of any other change management system, the team used Bugzilla to communicate changes in the development of particular work items.

As can be seen from Table 2, the amount of communication varies across developers quite substantially, with a low of 5 and a high of 92 interactions reported over the 19 day period. In terms of creating awareness mechanisms, this variability in amount of communication could have an impact. For someone who communicates frequently, a different type of awareness mechanism may be necessary than for someone who communicates infrequently. Furthermore, developers communicate quite frequently using Bugzilla which was not developed as a communication tool. This highlights the fact that developers will co-opt other tools to fit their needs, and this must be taken into account when searching for ways to provide awareness in a software engineering context.

The use of collaboration tools and mechanisms differed when used in local vs. remote communication. Face-to-face communication is only possible locally, and it was used for longer discussions. As shown in Chart 1, F2F communication represented the highest amount of interaction within the project (a total of 119 instances out of 418 (28%)).

If we add the proportion of F2F communication (28%) and phone communication (6%), a total of 34% of communication among the team members is neither recorded nor stored in a data repository. This poses challenging design requirements for awareness systems that use communication information in the creation of associations between people.

SameTime (11% of total), IBM's IRC tool, was used locally for quick questions or comments; in the remote communication it was used for longer conversation or to request a phone call. Email (18%) was used for both local and remote communication, and developer interview data suggests that email was used more for "requests that are not urgent", or "letting people know about something that has been done or will be done", or "initiating discussions that have some visual aspect, like code excerpts". Phone was almost never used locally.

Table 2. Number of interactions reported in the interaction log and postings on Bugzilla

Developer	Interactions reported in interaction log	Postings in Bugzilla
Brian - CA	92	20
Zoe - CA	58	6
Chris - CA	33	18
Lucas - CA	5	0
Trish - US	34	9
Joshua - US	23	2
Total	245	88

4.2.2. Evolving social networks in the collaborative development of WI 1009

As mentioned above, we tracked information on project members involved in the development of WI 1009 during the stages of its development life-cycle. Here we discuss the dynamic nature of the social network associated with this work item, emphasizing how the membership of this social network evolved throughout the development stages especially when compared to that initially planned for this work item.

Implementation Proposal Development and Review. As identified from the CSM component planning documentation, two Canadian developers only were assigned to create the implementation proposal for WI 1009, Zoe and Brian. Once a proposal is completed, it is posted to Bugzilla and reviewed by other team members during the *design phase*. Figure 1 shows a sociogram of the social network of the seven developers who contributed to the design of WI 1009 by posting comments. The directed arrows from person A to person B indicate that person A was the poster and person B the receiver respectively for the message. We observe two interesting patterns. First, for this particular work item, developers from additional US sites were involved in the discussion because they were clients of WI 1009 (their work was dependent on WI 1009).

Second, Brian and Zoe, who were assigned to implement WI 1009, do not only receive information from other developers, but also provide information to them.

We also see information exchanged among developers that are not Brian and Zoe. This indicates that the knowledge required to develop this work item is very distributed in nature, and is a mismatch from the plan.

Implementation and integration of WI 1009. After an agreement was reached on the implementation proposal, the coding began. Figure 2 shows the large number of project members that collaborated during the implementation of WI 1009. While three developers were the main coders (Brian, Zoe and Chris), the other developers were clients of the module containing the changes specified by WI 1009.

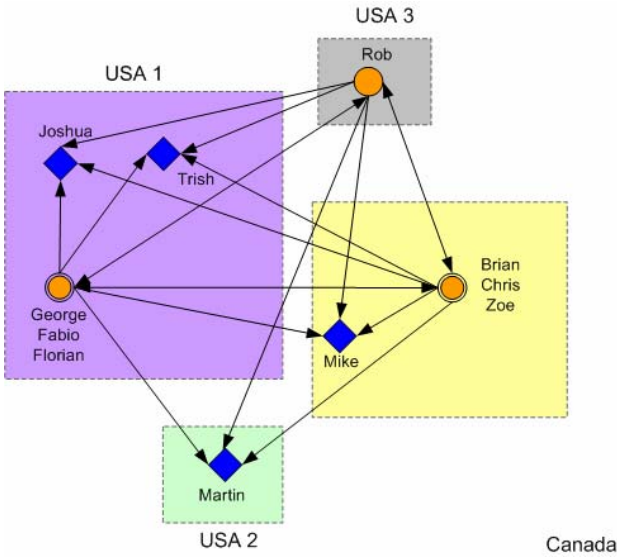

Figure 2. Communication exchanged over Bugzilla during design of WI 1009

Figure 2 also shows the communication data acquired from the developers' interaction logs during WI 1009's life. The numbers are summarized in Table 3. It is important to observe that the original model of implementation indicating just two members assigned to development has grown to a social network involving 17 members (developers and managers).

It is clear to see that not only do the social networks evolve as development occurs, but also that each phase of development has unique social networks. Moreover, the social networks that develop are almost completely unrelated to those that are documented in the project proposal. This information has important ramifications for awareness tools in distributed teams. First, the documentation does not reflect the actual character of the coordination necessary to implement software. Thus, any awareness system will need to find ways to determine who is relevant for which pieces of information independent of the documented team relationships. More interestingly, over only 19 days, the team not only evolved but changed dramatically. The team members involved in the design were almost entirely different than the team members involved in the implementation phase. The quickness with which the teams change over the course of a work item is quite astounding. A few central members are part of the whole development process, but other members come and go as the need for them arises and dissipates. Awareness mechanisms must be able to account for the dynamic nature of team collaboration and coordination throughout the development lifecycle. This is especially important when teams are globally distributed and the ability to track who is working with whom may be limited.

Table 3. Number of interactions regarding WI 1009 reported in interaction logs

Communication media	Number of times used
Face-to-face	51
Bugzilla	19
Email	13
SameTime	14
Phone	4

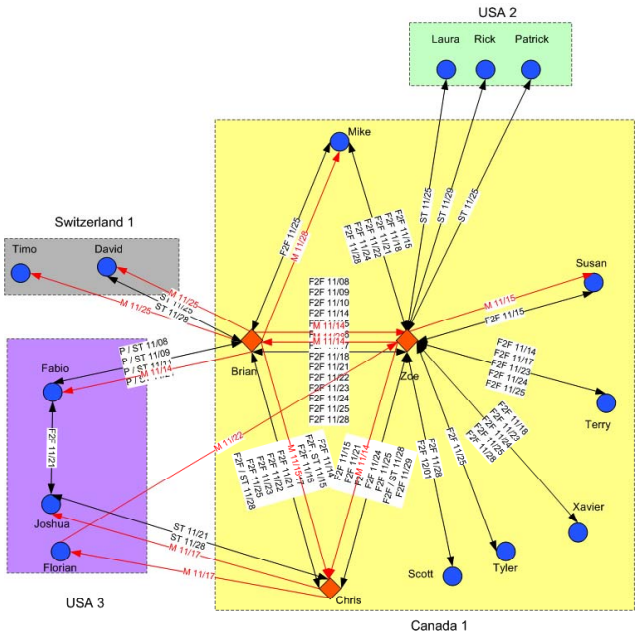

Figure 1. Interactions recorded in interaction logs during the implementation of WI 1009

4.3. Analysis of a Broken Build: Information Overload and a Lack of Awareness

Our further analysis of the communication within the dynamic social network and the amount of communication around the days of the broken build for WI 1009 reveals that the generation of the broken build was caused by a gap in communication between team members. Typically the generation of the weekly-integration build happens every Monday at night. We monitored both the continuous and the weekly integration builds for WI 1009 and, when the integration build was broken, we spoke with the developer responsible for the build in order to identify the reason why the broken build occurred.

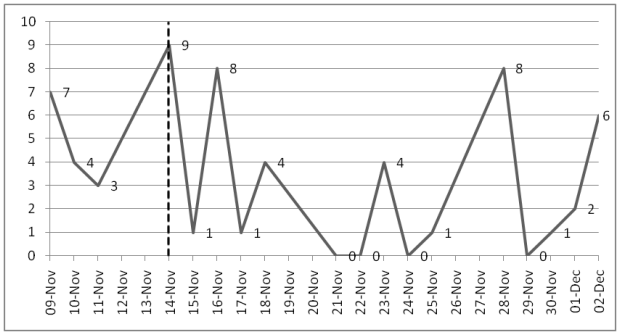

Chart 2. Number of messages on CSM mailing list by date

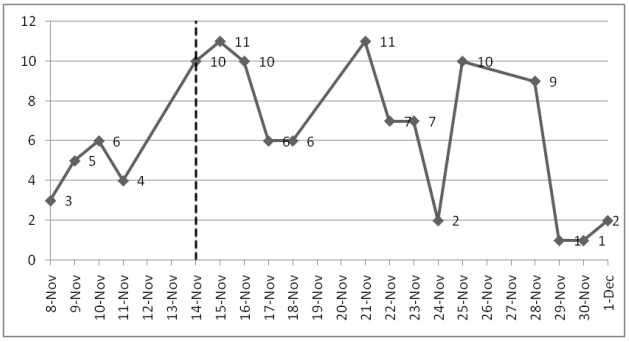

Chart 3. Number of face-to-face interactions reported by developers by date

Before the build was generated, the developer responsible for the preparation of the client submission of the integration build sent a notification email to every CSM developer and asked them to keep the related files in good shape throughout the day. When integration build occurred, the developer responsible for the build noticed that the failure of the build was originated by changes done by Zoe. Changes had to be rolled back to the point that allowed the generation of a green build, resulting in a loss of time and effort.

An interview with Zoe after the broken build indicated that due to the amount of information handled by her during the days of the integration build, she was not able to attend to the email sent by the build developer. In fact, she did not even read the email at all.

The examination of the communication patterns in the growing and dynamic social networks associated with WI 1009 provides some insights into this incident. Figure 2 shows the type of communication media, as reported in the interaction logs, used between the members of the social network, over the 19 day period. F2F indicates face-to-face interaction, P indicates phone conversation, ST refers to SameTime chat and M indicates email message. Note that this figure does not include messages that are broadcast on mailing lists.

4.3.1. Information Overload

Looking at Figure 2, one can then observe the significant network of interrelationships that should be handled by Zoe, which made her prone to miss information that could lead to the generation of a broken build. She was also indirectly related to the development of another work item in the CSM, and sometimes asked to provide support or share her expertise with other local and remote team members.

A closer examination of the volume of communication during the days of the generation of the broken build provides us with more insights about the amount of information that was handled by the developers of the distributed team. Chart 2 shows the number of messages posted in the CSM mailing list during the period observed. Chart 3 shows the amount of F2F interactions each day during this period, as reported by the six developers. The CSM mailing list is where the notification message, sent by Lucas in Figure 2, was distributed.

In these charts, one should observe the volume of communication generated on November 14th, the day when the broken build was caused. It can be seen that a high volume of mailing list traffic was generated on this day, the second highest this period in the project. With respect to the F2F interactions on November 14th, 4 out of the 10 reported involved Zoe. It is, however, reasonable to expect that the overall amount of communication was even higher, since these numbers only relate to *reported* interactions among only 6 developers, and the numbers of people on the social network implies a possibly larger amount of communication (including SameTime and individual email messages). We see that Zoe was subjected to a large volume of information on the day before the integration build scheduled that evening.

From this analysis, we conclude that in the particular time that the broken build occurred, the developers were exposed to a high volume of communication in the implementation of this WI in addition to supporting development in other components of the system. Handling this volume of information relied on the personal skills of the developers in tracking all the links or dependencies of their implementation. Zoe, in particular, had to handle a large amount of communication during a stressful period, and it is no surprise that she did not read the notification message sent by the developer responsible for the integration build.

4.3.2 Communication of changes in the development of WI 1009

To understand the communication of change information in the project, we explored how changes pertaining to the development of WI 1009 were communicated within these dynamic social networks. We collected data on the reason information was exchanged between team members shown in Figure 2. We sought to determine whether certain types of information were more predominantly communicated F2F, or other media such as email, phone or ST, and whether any interesting patterns with respect to communication of changes emerge.

Table 4. Classification of issues and number of interactions for each reported communication media. Percentage indicates proportion of media that is used for each category.

Category	F2F	Phone	SameTime	Email	Total Number
Implementation issues	**31 (36%)**	7 (8%)	21 (25%)	26 (31%)	**85** (100%)
Coordination of activities	**27 (41%)**	10 (15%)	15 (15%)	14 (21%)	66 (100%)
Communication of changes	9 (20%)	2 (4%)	8 (8%)	**26 (58%)**	45 (100%)
Synchronization	**22 (61%)**	5 (14%)	8 (8%)	1 (3%)	36 (100%)
Planning	**13 (65%)**	3 (15%)	1 (1%)	3 (15%)	20 (100%)
Support	**3 (18%)**	1 (6%)	10 (10%)	**3 (18%)**	17 (100%)
Risk Assessment	**1 (33%)**	0 (0%)	**1 (33%)**	**1 (33%)**	3 (100%)
Other - not relevant	0 (0%)	0 (0%)	0 (0%)	**1 (100%)**	1 (100%)

Table 4 summarizes the data we collected from the interaction logs. It can be observed that, overall, communication of changes was reported third in frequency, after coordination messages and implementation issues. Email was also observed to be the preferred medium when communicating changes; this is an exception to the dominant use of F2F interactions to discuss every other issue.

Change notification is important in a project, and Email is the preferred method of sending change notification. However, if developers are overloaded with Email messages and fail to read an important Email change notification, then they will be unable to react. We see in this case study how easy it is for a developer to miss critical notifications due to large volumes of communication.

5. Limitations of the research

The results of this research are limited by two primary considerations. First, the case study was conducted looking at only one project in one company. The results may or may not generalize to other projects, or other companies. While it appears safe to assume that this project is fairly representative of other projects in GSD, more research will need to be conducted to verify this claim. Second, the use of retrospective reports may limit the accurateness of the social networks. However, a more in-vivo way of reporting the social interactions would likely make the networks more complex, and not less so, which would support our results. Again, this finding will need to be confirmed using different data collection techniques.

6. Implications for research and practice

An understanding of the awareness needs of software engineers "in the wild" has not been studied yet. To remedy this situation, our study sought to empirically characterize aspects of awareness, and in particular what could be the factors leading to gaps in awareness, as well as the consequences of these gaps for coordination. We draw here some implications for further research and the development of improved awareness mechanisms.

First, our finding that organizational culture affected effective coordination of development and maintenance of awareness of work across development sites indicates that the causes of gaps in awareness are not simply technical in nature. Intra-organizational partnerships in global software development create working relationships in which factors such as culture play an important role and where new tools or processes are not a quick fix to the problem. Moreover, cross-site collaboration is affected not only by differences in the national culture of remote teams [14][13] but also in the way the organizational culture has developed and emerged across sites [1][11]. Therefore, organizational culture manifested in the way in which project teams follow processes or use common tools needs to be recognized as an important factor in affecting how awareness of change should be promoted between remote sites.

Second, our finding that the development of features involves a dynamic and evolving social network that does not reflect the plans set out in the initial task allocation has important implication for awareness systems. First, it corroborates with evidence that documentation quickly becomes out of date in organizations [19]. Thus, any awareness system will have to use other sources of information to understand the current status of the project. Second, a collaborative system should be able to facilitate unplanned collaborative work among software developers. Such a system must be able to (1) allow expertise finding such that these emerging interactions do occur, and (2) maintain awareness among each person who has contributed to the work item, without overloading them. One research question that we can draw from this finding is, "Who are these additional people, and why are they contributing, or why have they been contacted about the work item?" This finding also reveals that there is a significant amount of emergent interactions between developers across different geographical sites, despite the fact that these people are not mentioned in the plan. We know that a large amount of information is communicated face-to-face in a remote site, but what are the best ways to support informal communication among developers across sites? What knowledge is exchanged within a local site that makes a developer aware of experts in another geographical location? The fact the social network evolves to include many developers means that we should investigate mechanisms to keep these people aware and up-to-date with the work item.

Third, because of information overload due to subscription to mailing lists, a developer failed to read an important notification and checked in code that caused a broken build. The broken build directly led to productivity loss because changes had to be reversed, preventing other developers from building new code and testing changes until the build was repaired. It has been shown that maintaining

awareness using mailing lists requires significant effort on part of the developers [9], but we also see that developers transmit change notifications most frequently using email. One of our future research goals is to find a mechanism to filter notification messages such that they reach only recipients that need to be aware of the notification, thus reducing message volume and preventing awareness overload. By targeting developers specifically, we believe that the relevance of email messages will increase, and that notifications will be more effective.

This lack of awareness of the actual membership of the social network associated to the development of certain features is critical when teams fail to adequately communicate changes in the project. Our evidence of the relationship between information overload and broken integration builds has implications for coordination theories and supporting tools in software development and particularly in GSD. A study by Cataldo et al. [3] indicated that, in a study of coordination requirements, developers were not interacting with each other as stated in the plan. Our results indicate a different, but related result: that, as work on a work item progresses, the emergent team expands, and more people are involved in communication than was originally planned.

This study thus provides further empirical evidence about collaboration patterns in software development, and particularly with respect to aspects of awareness in distributed teams. When looked in the light of the sparse existing literature on awareness and coordination in software development, our work contributes to the development of very much needed theories of collaboration and coordination in software development.

7. References

[1] B. Berenbach. Impact of organizational structure on distributed requirements engineering processes: Lessons learned. In *GSD '06: 2006 International workshop on Global software development for the practitioner*, pages 15–19, New York, NY, USA, 2006. ACM Press.

[2] B. Bruegge, A.H. Dutoit, T. Wolf. Sysiphus: Enabling informal collaboration in global software development. In *First International Conference on Global Software Engineering, Florianópolis, Brazil,* October 16-19, 2006

[3] M. Cataldo, P. Wagstrom, J. Herbsleb, and K. Carley. Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools. In *Conference on Computer Supported Cooperative Work (CSCW'06), Banff, Alberta*, Canada, 2006.

[4] L.-T. Cheng, Susanne Hupfer, Steven Ross, John Patterson, Bryan Clark, Cleidson R. B. de Souza: Jazz: a collaborative application development environment. *OOPSLA Companion 2003*: pp. 102-103

[5] B. Curtis, H. Krasner, N. Iscoe: A Field Study of the Software Design Process for Large Systems. *Comm. of the ACM 31(11)*: 1268-1287 (1988)

[6] P. Dourish, V. Bellotti: Awareness and Coordination in Shared Workspaces. *CSCW 1992*: pp. 107-114

[7] S.B. Fonseca, C.R.B de Souza, D.F. Redmiles. Exploring the Relationship between Dependencies and Coordination to Support Global Software Development Projects. In *First International Conference on Global Software Engineering, Florianópolis, Brazil*, October 2006. pp 243-243.

[8] J. A. Espinosa, S. A. Slaughter, R. Kraut, and J. D. Herbsleb. "Team knowledge and coordination in geographically distributed software development". Journal of Management Information Systems, 24 (1), 2007. *To Appear.*

[9] C. Gutwin, R. Penner, K. A. Schneider: Group awareness in distributed software development. *CSCW 2004*: 72-81

[10] C. Gutwin, K. Schneider, R. Penner, and D. Paquette. Supporting Group Awareness in Distributed Software Development, *Engineering Human Computer Interaction and Interactive Systems, Revised Selected Papers, Springer–Verlag, Berlin*, 2005, pp. 383-397.

[11] C. A. Halverson, J. B. Ellis, C. Danis, and W. A. Kellogg. Designing task visualizations to support the coordination of work in software development. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 39–48, New York, NY, USA, 2006. ACM Press.

[12] J. D. Herbsleb, R. E. Grinter: Architectures, Coordination, and Distance: Conway's Law and Beyond. *IEEE Software 16(5)*: 63-70 (1999)

[13] G. Hofstede. Culture's Consequences: Comparing Values, Behaviours, Institutions, and Organizations Across Nations. *Sage Publications: Thousand Oaks, California*. 2001.

[14] Y. Hsieh. Culture and Shared Understanding in Distributed Requirements Engineering. In *First International Conference on Global Software Engineering, Florianópolis, Brazil*, October 2006. pp. 101-108.

[15] T. W. Malone and K. Crowston. "The Interdisciplinary Study of Coordination." *Computing Surveys*, 26 (1), 1994.

[16] A. Sarma, Z. Noroozi, André van der Hoek: Palantír: Raising Awareness among Configuration Management Workspaces . *ICSE 2003*: pp. 444-454

[17] A. Sarma and A. van der Hoek, Towards Awareness in the Large. In *First International Conference on Global Software Engineering, Florianópolis, Brazil,* October 2006, pages 127-131

[18] T. Schümmer: Lost and Found in Software Space. *HICSS 2001*

[19] J. Singer: Practices of Software Maintenance. In *International Conference on Software Maintenance (ICSM) 1998*: pp. 139-145