To Talk or Not to Talk: Factors that Influence Communication around Changesets

Adrian Schröter
University of Victoria, Canada
schadr@acm.org

Jorge Aranda
University of Victoria, Canada
jaranda@uvic.ca

Daniela Damian
University of Victoria, Canada
danielad@cs.uvic.ca

ABSTRACT

Building tools to help software developers communicate effectively requires a deep understanding of their communication dynamics. To date we do not have good comprehension of why developers talk to each other as a result of *some* events in the life of their projects, and not of others. This lack of knowledge makes it difficult to design useful communication models and support systems.

In this paper, we narrow down the study of communication behaviour to focus on interactions that occur as a result of a particular kind of project event: the submission of a changeset to the project repository. In a case study with the IBM® Rational® Team Concert™ development team we investigate which factors influence developers to request information about a changeset to their product. We identify several such factors, including the development mode in which the team is operating, the background and recent performance of the author of the changeset, and the risk that the changeset poses to the stability of the product. Incorporating these factors into communication support systems and failure prediction models may lead to improvements in their performance.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Programming teams*; K.6.3 [**Management of Computing and Information Systems**]: Software Management—*Software development*

General Terms

Human Factors,Management

Keywords

Information Sharing, Communication, Software Teams

1. INTRODUCTION

For all but the most trivial programs, software development today is a collaborative activity. The members of software teams must coordinate and communicate, often intensely, if their projects are to reach a satisfactory conclusion. Previous research has shown that communication among developers has a profound influence on several aspects of software development [15, 25].

Despite this acknowledged importance of communication in software teams, our understanding of communication behaviour is still very limited. Ko *et al.* [17] point out that developers often require information about the work output of their peers. We know little about why developers act upon this requirement. What causes developers to seek information in some cases, but not in others? There are a multitude of potential factors: the closeness of their peers' work with their own, the expertise of the developers, their physical proximity, their level of workload, and even, perhaps, the time of day. By identifying which factors actually influence the likelihood that a developer will request information about the events surrounding her, we might better understand the nature of software-centric communication, leading to better practical applications for our research.

For this study we decided to investigate communication revolving around changesets, since a study of changesets can be generic enough to be applicable in many settings, but particular enough to be informative about detailed patterns of behaviour. We report on a field study that used a mixed-methods approach (participant-observation, surveys, and interviews) to understand what technical, social, and process-based factors influence whether a developer will seek information about a changeset or not. We investigated communication in the IBM Rational Team Concert (RTC) project, which is distributed across North America, Europe, and Asia, and in which communication is linked to changeset data. The RTC project presents unique opportunities for research on communication in software teams not only because it takes place within a development environment in which a significant fraction of communication is recorded in technical artifacts, but also because the team is tasked with the goal of developing these artifacts themselves. The challenge of effective team communication is particularly present in the RTC project team.

In the remainder of the paper we present the motivation and background for our study, and then describe our study design. We present in detail the results of our analysis of the data, and distill a number of technical, process and social factors that we identified as influencing developers' information-seeking behaviour. We conclude by discussing the implications of our findings on the design of collaborative systems.

2. BACKGROUND AND MOTIVATION

This research builds upon the growing body of work that studies coordination and communication in software organizations, which have long been recognized as playing central roles in leading software projects to success, e.g. [8, 18].

Our research group gained the opportunity to study the IBM Rational Team Concert (RTC) development team and the software artifacts produced by this team over several years. The RTC team maintains and cross-links all of their software and process artifacts using the very tool they produce. Partly due to its distributed nature, and partly due to its open source origins, the team's emphasis on written and archivable electronic communication results in a wealth of data for analysis.

Our previous investigations of the RTC project repositories revealed a significant relationship between gaps in communication structures and build success [25]. Therefore we proposed a recommendation system that allows team members to learn about who they should talk with in order to avoid build failures [23]. This recommendation system should focus on changesets and analyze significant differences between code dependencies and team communication structure to identify anti-patterns that the developers should attempt to break. The basis for this principle is derived from Conway's Law [7] and by the model of socio-technical congruence proposed by Cataldo *et al.* [5, 6]. The model compares the communication network to the code dependency network and derives insights with respect to productivity from the gaps between those networks.

In order to build a good recommendation system, however, one needs to go beyond providing valuable information. As Murphy and Murphy-Hill show [20], we need developers to trust the recommender system: delivering the right recommendation at the right time is one important way to build such trust. Unfortunately, recommender systems often fall short on this front, saturating the developer with recommendations they neither need nor want, or presenting recommendations at inappropriate times. It is therefore important to know when developers actually need information, and what factors influence them to seek information. There are many potential factors that could change the likelihood that developers would want to obtain further information, and we do not have a sense, other than our intuition, to learn which factors are actually important and which are not.

We are not the first to study factors that influence communication among software developers. For instance, while APIs are intended to reduce coordination complexity among development teams, they also introduce barriers that hinder communication between teams by obscuring dependencies [9, 10]. Software architecture, organizational structure and project age are also factors that influence communication and the awareness that developers have of each other's work [11]. Processes, in particular, are often used to influence the communication among team members to reduce information overload [13], and geographic distance has been shown to affect communication, but its effects vary between projects [14, 24]. Nakakoji *et al.* [22] distilled the current knowledge about communication among developers into a set of guidelines to design tools and processes that facilitate effective expertise communication.

Those studies, and to our knowledge, literature in software engineering in general, only address communication on a project-wide level. However, to fully understand, and thus offer actionable knowledge to developers, we need to know what influences an individual developer's information-seeking behaviour. We surveyed literature to compile a list of possible information-seeking triggers spanning change-related (such as code metrics), developer-related (such as experience level) and process-related (such as peer reviews) factors.

Therefore we pursue to answer the following **research question:** *What factors influence whether a developer will decide to seek information about a changeset in her project?*

3. STUDY DESIGN

To address our research question, we complement our previous analysis of the RTC electronic repositories with rich qualitative data from interviews and observations, considering that much information on a project's lifecycle is not recorded in its repositories [1]. We focused on studying what factors cause a developer to seek information about a changeset, since changesets are the smallest units of change to a project that directly impact other developers.

Our research question called for a mixed methods empirical study. Therefore, we collected data about the team's information-seeking behaviour from three different sources:

1. The first author of this paper (Adrian) was embedded in a subteam of the RTC team as a participant-observer for four months.

2. We deployed a survey with the entire development team to validate some of the findings from the observations.

3. We conducted interviews with the developers from one component team to obtain richer information about their communication behaviour.

3.1 Study Setting

Rational Team Concert is a scalable and extensible team collaboration platform that integrates source code management, issue tracking, and planning tools. The RTC platform uses a client-server architecture, with the server acting as a central data repository. The team also uses RTC to develop RTC, and in this way RTC is a self-hosting project. This helps the RTC team become aware of issues with the product before customers.The project follows a large scale iterative development process that emphasizes transparency [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 each iteration, and the output of this process is assigned to the teams in the form of work items. The work items and nearly all documentation are publicly available.

The RTC development team has over 100 developers distributed across multiple sites in North America, Europe and Asia. The RTC development team is subdivided into several teams that manage a component each, such as the source code system or the web user interface. To communicate with each other, especially across different geographical locations, the developers extensively use RTC's own issue and task management system. A particular characteristic of the RTC development environment—and one that made this study of communication behaviour around changesets possible—are explicit traceable links between work tasks, associated changesets, and developers' online comments about these tasks and changesets[1].

3.2 Data Collection

We used a mixed methods approach to answer our research question. We obtained data from three sources: participant observation, a survey, and a set of interviews with RTC team members.

[1]Changesets in RTC are created in a developer workspace; each changeset consists of several changes to code files.

3.2.1 Participant-Observation

The first author of this paper (Adrian) joined one of the RTC development sites in Europe for a four-month period in the Fall of 2010. He was involved as an intern with this development team, and thus was in a position to directly participate in the project development and communication activities. Our data collection opportunities were thus much richer than in typical observations conducted over shorter periods of time, or that do not involve active participation in the project. During his period as a participant-observer, Adrian kept a daily activity log, recording any information of relevance to the communication behaviour of his team.

The observation period coincided with the months prior to a major release and during which the team focused on extensive testing rather than new feature development. As such, the majority of our observations are concerned with activities around testing the integration of RTC with other IBM Rational products, and it offered the experience of collaborating with many developers from different teams and across the remote sites. Finally, in the one month following the release, Adrian also participated in the development of a feature for the upcoming version of the product.

The team in which he was embedded had the responsibility to develop the task management and the planning components of RTC. There were ten developers at the site, including two novices that had recently joined the team and eight experienced developers. The team includes RTC's three senior project management members, who play a major role in planning the product's architecture and development.

3.2.2 Interviews

We conducted interviews with ten developers of the RTC team to inform our observational findings. The setting for the interviews was slightly unusual, and beneficial from a research perspective, since the first author's participant-observation allowed him to develop working relationships during his stay at the team, and to delve into a discussion of complex communication issues without needing to spend time understanding the basic context of the team.

In the interviews, we requested developers to provide details into their communication dynamics through a narration of "war stories" that directly related to communication among developers. "*War stories*" are events that left an impression on the respondent. War stories emphasize specific events that they witnessed or took part in [19]; their narrative is particularly powerful at uncovering the elements of the stories that are relevant to the respondents. The interviews lasted between thirty minutes and one hour, and they were conducted at the end of the observation period.

3.2.3 Survey

To complement the insights we obtained from the observations and interviews, we deployed a survey with the entire RTC team at the end of our observation period. The survey was designed iteratively and piloted with a European team, and intended to collect input about which factors increase the likelihood of a developer requesting information about a changeset from the changeset author. The items we included in the survey were in the categories of code-related, developer-related and process-related items. Each category included 14 items, as shown in Table 1. *Code-change related* items describe the changeset delivered or the code that the changeset modifies or affects. *Developer-related* items relate to the developers that delivered a changeset, the developer requesting the information, and their relation to each other. Finally, *process-related* items relate to the development phase in effect when the changeset was delivered, and to items relating to process requirements or practices. For each of the three categories, the survey asked the developers to rank each item in the category according to how strongly it increases their likelihood of requesting information about a changeset.

We initially formulated our list of items from an analysis of the current literature on coordination and communication, as well as from our own previous research. We then refined the list of items through piloting and discussions with the development team. An initial version of the survey included fifty-nine items; the refined list had forty-two. Besides reducing the number of items in our list, the pilot survey led us to add several process-related items, as well as to change the format of the question (the initial survey had a battery of questions with Likert-scale answers; the final survey asked respondents to rank each item in comparison to the rest in its own category).

We deployed the survey through a web form, and invited the entire distributed RTC development team to participate. We sent a reminder one month later to increase the response rate. We obtained 36 responses to the survey; approximately a 25% response rate.

3.3 Analysis

For our data analysis, we transcribed the interviews, and then coded both the interviews and the participant-observation notes. From the codes we created categories, which we cross-referenced to the survey rankings. We used an open coding approach, during which we assigned codes to summarize incidents in sentences and paragraphs of the transcribed interviews that relate to our research question.

We calculated the ranking of the survey items for each category with a points-based system in which we added the reverse ranks given by each developer. For instance, if a developer ranks an item to be first in its category, then it is worth 14 points (each category has 14 items). In case of changes to the API, the first item in Table 1a received 390 points which is the sum of nine developers ranking it first ($9 \times 14 = 126$ points), eleven developers ranking it second (143 points), two developer ranking it third (24 points), four developers ranking it fourth, four fifth, and one sixth, seventh and ninth each. Four developers did not include this item in their rankings.

During our analysis, we derived a set of factors affecting communication behaviour based on our list of survey items and qualitative data from our interviews and observations. Whenever possible, we triangulated our findings using all our data sources. We tried to ensure that all of our findings were supported by at least two data sources; none of the findings we report were challenged by any of our data sources.

Table 1 summarizes the ranking that each item received from the entire set of individual responses to our survey. Next to each item, we indicate, whenever possible, quotes from our interviews and observations that corroborate the position of the item in the list.

4. FINDINGS

The evidence collected from our survey, interviews, and participant observation allowed us to derive seven factors that affect communication behaviour in software organizations, which we outline below. In the remainder of this section, text in italics refers to survey items that can be found in Table 1. Text in italics and around quotation marks denotes interview fragments.

Survey Items	Interview Quotes (I)/Participant Observation Quotes (O)	Points
Changed API	(O) *"Adrian, why did you change that API?"* [The RTC team avoids changing API.]	390
Don't know why code was changed	(O) *"I don't see a reason why you changed that code, you sure you needed to?"*	348
Affects frequently used features	(I) *"Before I ok a fix [even to a main feature], I ask if there is a workaround."*	325
Complex code	—	289
Introduced new functionality	—	260
Is used by many other methods	(O) *"Why did you fix this part? The bug is not in this part of the code, it is used and tested a lot."*	240
Your code was changed	(I) *"I'd be very irritated if someone other than [...] would touch my code."*	232
Stable code was changed	(O) *"Why did you fix this part? The bug is not in this part of the code, it is used and tested a lot."*	213
Change unlocks previously unused code	—	192
A bug fix	—	162
A re-factoring	(O) *"I separate a re-factoring from a fix so that people can ask me questions about the fix."*	152
Frequently modified code was changed	(I) *"I like to know where the 'construction sites' of the project are."*	118
Code is used by few other methods	—	94
Simple code was changed	—	58

(a) Code-change-related items and quotes

Survey Items	Interview Quotes (I)/Participant Observation Quotes (O)	Points
Author is inexperienced	(O) *"Adrian, please put me always as a reviewer on you change-sets."*	385
Author recently delivered sub-standard work	(I) *"She just changed teams and still needs to get used to this component."*	335
Author is not up to date with recent events	(I) *"After you return from vacation we ensure you follow new decisions."*	329
You do not know the change-set author	(I) *"I'd be very irritated if someone other than [...] would touch my code."*	269
Author is currently working with you	(O) *"We sometimes discuss changes we made to brag about a cool hack."*	226
Author part of same feature team	—	190
Author part of your team	—	188
Worked with author before	—	165
Busyness of yourself	(I) *"If I see a problematic change-set, I'll ask for clarifications even if I'm busy."*	164
Busyness of the author	(I) *"If I need to know why an author made that change I just contact him." [Even if he is busy.]*	147
Met in person	(I) *"I've worked with him for 5 years now but never even met him."*	117
Physical location	(O) *"Here, America or Asia, I don't care, I ping them when they are online."*	112
Author is experienced	—	93
Author recently delivered high-quality work	—	86

(b) Developer-related items and quotes

Survey Items	Interview Quotes (I)/Participant Observation Quotes (O)	Points
Release endgame	(O) *"Adrian, in the endgame we only do minimal changes. Is your change minimal?"*	319
To review the change	(I) *"I often lack sufficient understanding of the part of the code I'm reviewing."*	311
To approve the change	—	298
Late milestone	—	293
During iteration endgame	(O) *"Adrian, in the endgame we only do minimal changes. Is your change minimal?"*	288
Obtain a review for the change	(O) *"It is demotivating to get a reject, thus I talk to my reviewer beforehand."*	287
Obtain an approval for the change	(O) *"I already fixed it, but I need to convince my team lead to give me approval."*	281
Related work item has high severity	—	246
Verify a fix	(I) *"Often I need to ask how I can tell that a change-set actually fixed the bug."*	235
Topic of work item the change is attached to	—	181
Priority set by your team lead	—	152
Role of the committer (e.g. developer)	—	145
Early in an iteration	(I) *"... there are weeks I sometimes don't talk to my colleagues at all."'*	66
Early milestone	—	60

(c) Process-related items and quotes

Table 1: This table (in three parts) summarizes the data we collected from our three data sources: 1.) our participant-observer's log-book; 2.) survey responses from 36 developers situated around the world; 3.) interviews with 10 developers. These items are ranked by points, which represent the cumulative importance the respondents assigned to each item in the survey. Whenever possible, we provided quotes from interviews (I), or quotes from the participant-observation period (O) to exemplify the items.

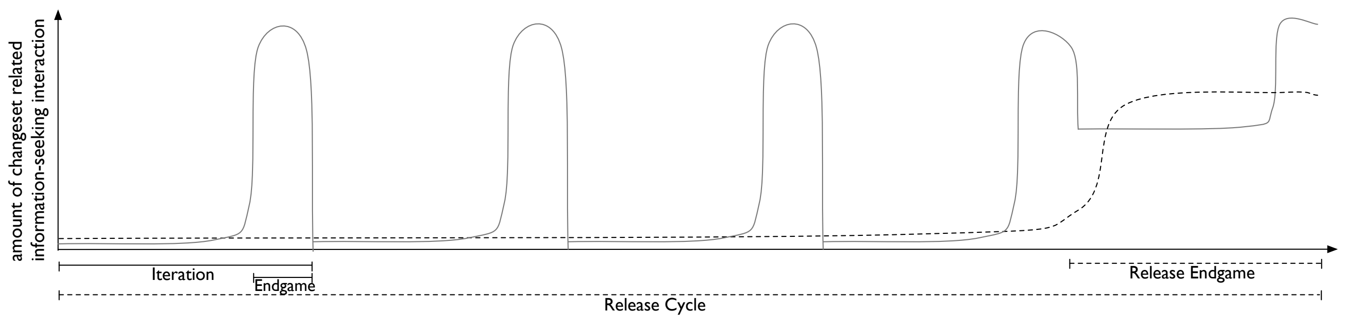

Figure 1: The pattern of information-seeking interactions throughout several iterations of a release cycle. Every release cycle consists of a number of iterations; each iteration includes an endgame phase. Changeset-based interactions are more frequent during endgame phases and during the last iteration of the release cycle.

4.1 Development Mode

One of the strongest findings from our study was the effect of the development mode that the team is in on its communication behaviour. Developers and managers alike give much importance to the development mode they are in, namely (1) normal iteration, and (2) endgame.

The normal iteration mode mainly consists of work that can be planned by the developer. This work tends to be new feature development, or modifications to existing features. Most of the planning for it has been laid out in advance; furthermore, each developer individually knows what features have been assigned to her, and can plan ahead to meet her obligations. In contrast, the endgame mode mainly consists of work that is coming uncontrollably in short intervals from others. As a result of integration and more intense testing, defects crop up more rapidly and need to be addressed more quickly. Beyond allocating time for endgame activities, there is very little in this development mode that can be planned in advance.

The RTC team switches between the two modes in each iteration. Of the six weeks of a regular iteration, four weeks are assigned to normal iteration mode and two weeks to endgame. However, as deadlines approach, there are occasional special iterations that consist mostly of endgame-like work. In this manner, the same detailed pattern of alternation between normal iteration and endgame within an iteration is reproduced at a higher level, as shown in Figure 1.

We identified the same pattern with respect to the amount of changeset-related interactions in the team. We found that the mode in which the team is currently in is an important factor determining whether developers will feel a need to communicate about changesets. Being in endgame mode increases the need for developers to communicate about a changeset, whereas being in normal mode decreases the need to request information about it.

The first author of this paper (Adrian) experienced both modes during his participant-observer time with the RTC team. When he joined the team, it was in a release endgame phase, which was characterized by fixing bugs that were reported by testers. When the team released the project, it switched gears, and in the decompression after the endgame he had the opportunity to develop a feature for the product. The differences in the patterns of interaction between the two modes were distinctly clear, for him and for his peers. During the endgame mode, developers were essentially "on demand," available to fix whatever bugs were discovered. Later, during the normal iteration mode, they experienced far more autonomy and control over their time, and began working on activities that were less demanding of an interaction back-and-forth, and especially less demanding of keeping track of the changesets committed by their peers. In our interviews, developers almost unanimously pointed to the separation of the two types of modes, describing the differences in the type of communication in each, and identifying the autonomous *vs.* uncontrollable, inside *vs.* outside influence.

This is not to say that developers do not communicate while they are in normal iteration mode, but that the nature of their communication seems to be different. In normal iteration mode, developers communicate less about concrete changesets, but they communicate more about high level ideas: for instance, they raise questions about how a feature fits into the existing architecture or general tactics to implement a feature, about how much of it actually needs to be implemented or can be reused from other libraries, and so forth. And as a developer commented in one of our interviews, *"during feature development iterations there are weeks I sometimes don't talk to my colleagues at all."*

In contrast, the endgame mode is characterized by bug reports and last minute feature requests—that is, by work coming into every developer's desk uncontrollably. changesets become first class concerns during this mode because each change threatens the stability of the product: developers need to evaluate whether the changeset actually improves the product instead of making things worse. Therefore, they develop a habit of reviewing each other's changesets to assess the risk they pose to the project's stability.

Our survey data provides significant support for this finding. In Table 1c, we can see that overall, for the RTC development team at large, there is a greater likelihood to request information during product-stabilizing phases, such as *during the release endgame*, and a lesser likelihood to request information when working towards an *early milestone*. The former was ranked as the most important of the process-related items in our survey; the latter was ranked as the least important. All the survey items that refer to a late stage in the iteration or in the project lifecycle were ranked highly, and all the items that refer to an early stage were ranked lowly. This finding resonates especially with the team we interviewed.

The likelihood that a developer will request information about a changeset increases when the team is in a development mode characterized by uncontrollable work coming from outside.

4.2 Perceived Knowledge of the changeset Author

One key determinant for a developer to seek information about a changeset consists of what the developer knows about the background of its author. Several aspects of the author's background seem to be important: the quality of her recent work, her level of experience, and her awareness of recent team events or decisions.

First, these factors are important in part because they point to potential problems with the stability of the product. If a changeset author has *recently delivered sub-standard work*, other developers will want to ensure that the new changeset will not deteriorate their product, and therefore they will try to find more information about it.

Similarly, when the author is *inexperienced*, or is not *up to date* with recent team events for any reason, the risk of introducing problems into the product increases. One developer that interacts frequently with a newly founded team told us that *"most of the work that I need to review is of very low quality and needs several iterations before it is up to our standards."* Novices, generally speaking, face a ramp-up problem that takes a significant amount of time to overcome [3]. This was the case with Adrian's work, too: his first changesets were subjected to more scrutiny due to his unfamiliarity with the code base.

The survey data backs up these observations. The three top items in Table 1b consist of factors related to an unfavourable perception of the background of the changeset author. In comparison, the two least important items in that table refer to favourable perceptions of said background.

In essence, these items point to the relevance of trust in software development teams. For developers, trust in the skills of their colleagues is important. If this trust is non-existent (for instance, if a developer does not know the changeset author), developers are more likely to check the changeset and request information to ensure that the modifications were truly necessary and appropriate. But when trust exists (that is, when the author has a good record of delivering high-quality work, or is known to be an experienced team member), the need to seek information about a changeset decreases.

It is more likely that a developer will request information about a changeset delivered from an author that recently delivered sub-standard work, or is perceived as inexperienced or unaware of team events and decisions.

4.3 Common Experience and Location

Shared work experiences affect the likelihood of information-seeking behaviour in the RTC distributed team. This is partly related to the trust and perception issues discussed above, but also to the establishment of interpersonal relations and to the intertwining work responsibilities and expectations.

According to our interviews, this is the case particularly for senior team members. Although they work in a globally distributed team, they have managed to establish personal relationships with developers at many different sites. Through continuous interaction and through social events, they have learned more about each other, and thus feel more comfortable to initiate contact with them. Junior team members, in contrast, know very few of their teammates globally, and since they simply do not have these interpersonal connections with the rest of their team, it is less important to them whether they know the author of a changeset when needing to contact them.

Generally speaking, despite the lack of opportunities to build personal relationships with off-site people, for most developers sharing common experiences makes it easier to contact teammates to request information. One team lead told us: *"I have one or two contact people in each team we usually work with whom I ask for information and then often enough get referred to the right person in their team."*

Our participant-observation data confirm this. At one point, Adrian needed to set up and test a specific component in the RTC product. The development team in charge of that component was located on a different site. However, since Adrian had previously had the opportunity to establish relationships with some of the developers of that team through a previous research study, it was easier for him to contact them and ask for help in his setup task.

Our survey data corroborate these observations. It suggests that the developers that form relationships (on a personal or work level) with other developers gave more emphasis on previous experiences with them. For instance, it made a difference whether the developers *are currently working together* (Table 1b). Additionally, whether they have shared work experiences seems to be a more determinant factor than having *met in person* or sharing the same *physical location*.

This last finding would seem to indicate that the RTC team has managed to overcome obstacles created by geographical distance. Developers state that the physical location of a changeset author is not an important factor to seek more information about the changeset. Nevertheless, we should note two things. First, this finding merely points out that developers have no greater need to inform themselves of work output of their remote peers than of their local peers; it says nothing about the ease with which such information is acquired. Second, the RTC team has determined to carry out as many of its interactions as possible through textual, electronic media. This neglects the natural advantages provided by proximity in favour of uniform accessibility of interactions, no matter the physical location of the interlocutors.

Recent or shared work experiences eases the act of requesting information from distributed or local developers.

4.4 Risk Assessment

A central factor at the heart of the decision to seek information is a concern with the quality of the product and components developed by the team at large. Both managers and developers share this concern. Because of it, every team member is constantly evaluating whether there are significant risks involved in accepting a changeset and including it into the final product. This concern is greatest close to releases or major milestones, and less important when the team is in normal iteration mode.

Developers request information more frequently about changesets that touch on code that has a high customer impact. According to our interviews and observations, code parts with a high customer impact are those that are directly related to frequently used features or changes to the API that might be used by customers to customize the RTC product. In the case of the impact of API changes, there is an extensive knowledge exchange in the jazz.net forums between customers and developers about how to use the API to build extensions to RTC, which gives developers plenty of information about the possible impact of API changes to the customers. Consequently, changesets that modify code that has less impact on the customer are of less concern and less likely to be discussed. For example, when scanning fixes to reported bugs, developers perform a risk assessment to determine whether the bug fix is even needed. In the words of one team member, *"with every tenth bug fix you introduce another bug to the system, so unless the bug does not have a workaround and is something most customers would experience, we give it low priority."*

The ranking of items in Table 1a provides additional insights into the kinds of changesets that carry a risk for developers. A *changed API* comes at the top of the list, code that *affects frequently used* *features* comes third, immediately followed by changes to *complex code*, changes that *introduced new functionality*, and changes to code that *is used by many other methods*.

Developers investigate more carefully those changesets that might have a high impact to the customers or to the stability of the code base, and thus they are more likely to request information about them.

4.5 Work Allocation and Peer Reviews

In the later stages of the development cycle, developers have two process-based constraints that prompt them to interact with their peers. First, they need their peers to review their changesets before they are included in the product. Second, they need an approval to start working on a task (a triaging process needs to take place so that developers are allocated to the most important work items).

In the changeset review process, there are two ways the changeset author and the reviewers interact. Either the changeset author submits his changeset for review, or the changeset author discusses the changeset with the reviewer before submitting it for review. The main discriminant between meeting with the reviewer to discuss the changeset before submitting it or submitting without prior discussion seems to be whether the reviewer and the author are co-located.

Adrian's mentor during his time with the RTC team explained: *"It is very demotivating to get a changeset rejected, that's why we usually try to discuss thing before and then the reviewing just becomes all about quickly testing and accepting the changeset."* Of course, this does not prevent an already discussed changeset from being rejected, but it signals that the process-based constraint is not as strict as to forbid discussions about a changeset between authors and reviewers in the interest of an unbiased code review.

In the case of developers seeking approval to start working on a task, the process is similarly flexible. The approval is supposed to be issued before the developer begins work on the task. However, we observed several times that developers performed quick fixes, and later went to their team leads to acquire approval for the fixes they had already performed.

Several items in our survey point to the importance of information-seeking around the processes of approval and peer review. In Table 1c, the second- and third-highest items were information needed because the respondent needed *to review the change* or *to approve the change*. The need to *obtain a review for the change* or to *obtain an approval for the change* were of moderate importance. *Verifying a fix* was not judged as important by our respondents.

Peer review processes (needing to review or to have one's work reviewed) and approval processes (requiring an approval or granting an approval) trigger people to discuss changesets.

4.6 Business Goals vs Developers' Pride

Senior team members (managers and team leads among them) often ask about the purpose and the real need of a changeset in order to minimize unpleasant surprises that might be introduced with further code changes, and simply to have an economically feasible product. One senior development lead told us: *"often I need to stop my developers from fixing every little bug otherwise we would never be able to ship."*

Although every team member is concerned with the quality of their product, a more realistic, business-oriented perspective seems to be more prominent in the senior team members' information-seeking behaviour: they may request clarifications on the business case of changesets or work items if they are not immediately clear in the existing documentation.

More junior team members, however, do not have this concern. They express their pride in the actual product and want it to be as bug free as possible. A developer commented to us: *"I want to be proud of what I deliver and I am fairly certain if we don't fix it now it will come back to haunt us because it might upset some customers."* Not only does this lead the developer to try to convince her manager that including changesets for seemingly unimportant fixes is valuable, it also makes her information-seeking behaviour different with respect to that of the more senior team members. Broadly speaking, managers will be more interested in seeking clarifications as to the business case of a work item if it is not clear in its description, while developers will not care as much about this.

Senior team members discuss changesets when their importance to the product's release is not evident in their archives.

4.7 Type of Change

The three main reasons for a developer to commit a changeset are to (1) do feature development, (2) fix bugs, and (3) refactor. Among them, feature development seems to be the one that prompts developers to seek information the most. In our survey (Table 1c), code that *introduced new functionality* is ranked fifth among the code-change related factors, while changes that consist of *a bug fix* and *a refactoring* ranked far lower on the same list.

This may be due to the fact that feature development has less tool support than bug fixing (which is supported by debugging tools and automated test frameworks) and refactoring (which is essentially an automatic process today). The limited tool support and the inherently difficult nature of the task itself makes changesets that introduce new functionality more difficult to understand or assess. Hence, developers are more likely to request information about them.

Among all types of changesets, those that revolve around feature development are most likely to make developers request information about them.

5. DISCUSSION

Previous research has made it clear that developers communicate frequently to satisfy a number of information needs. However, in order to help them communicate and perform more effectively, we need better models of their communication behaviour. More specifically, we know that developers need to stay abreast of the work committed by their peers, but we do not know which, among the many pieces of work their peers commit, they actually care to know more about, and which do they feel they can safely ignore.

In this study, we provide some answers to this question by uncovering a number of factors that influence the information-seeking behaviour of developers. Our point of origin was the fact that developers often need to learn more information about the changesets committed by their colleagues. We aimed to discover the factors that trigger them to look for further information when they do.

One major factor that influences developers' information-seeking behaviour is the development mode in which their team is in. Developers differentiate between two main development modes: normal iteration mode and endgame mode. These modes differ deeply by the kind of discussion that developers have about their products, and by the amount of control and autonomy they have over their activities. The tools and heuristics that work for one are not likely to work for the other. Fortunately for researchers and tool designers, at least for the RTC team it is usually clear which mode the team is on, as the transition between normal iteration and endgame modes is made explicitly. Therefore, this is a finding that can be exploited with relative ease in models and tools that support developer information-seeking behaviour.

Other findings from our study are also directly applicable in practice. Communication behaviour is at least partly determined by the organizational process, such as requests to perform code reviews and requests for approval to address certain work items during endgame; these communication patterns can be understood and modelled. Additionally, developers consistently reported that they request information about certain tasks more frequently than others. They tend to be less likely to ask about routine re-factorings or bug fixes than about changes introducing new features or functionality. Bug fixes and re-factorings do not change the expected behaviour of a product, while new functionality introduces risk to the product; therefore it requires additional scrutiny.

Risk assessment, in fact, guides many developer decisions on whether to engage or interact with others to discuss changes made to the product. For example, changesets that affect many parts of the code (such as changes to the API) raise flags more often than changes which are fairly isolated. But there are two kinds of risks—the immediate risk that the product contains failures, and the long-term risk that a certain change increases maintenance in the future or takes the product in the wrong direction. This balance is hard to strike. Broadly, team members in junior positions seem concerned with the former, and those in more managerial positions (senior developers and team leads) with the latter; their information-seeking patterns vary accordingly.

Trust arose as a major issue throughout our findings. The changeset author's background affects whether a developer will seek further information on the change itself: if the developer trusts the author (that is, if they perceive that the author has the skills necessary to implement the relevant changes with little assistance), she will feel a lesser need to oversee their work. At the same time, however, if two developers have shared work experiences, the threshold to establish communication between them decreases. In effect, this means that two developers may communicate more often about changesets even if none of them authored them: for instance, one developer could be acting as a bridge or a connector between the other developer and the right person to discuss a changeset in a different team. This is a practice commonly found in distributed teams [24].

5.1 Recommender System Design Guidelines

Nakakoji *et al.* [22] formulated nine design guidelines for systems supporting information-seeking in software teams. Some of them deal with minimizing the interruptions experienced by the developers who are asked for information, while others refer to enabling the information-seeker to contact the right people. Our findings help us refine two guidelines from Nakakoji *et al.* that specifically refer to integrating the expertise communication with developer activities and to contextualizing the information it provides. We offer the following four design guidelines:

Guideline #1: *Communication support systems should adjust to the development mode.* The information-seeking behaviour of developers changes drastically with their development mode. When in normal iteration mode, developers act upon planned work and can therefore anticipate the information they need. Tools from which developers pull information, such as Codebook [2], support this kind of information-seeking behaviour, whereas tools that create a stream of information from events as they happen (like Ensemble [26]) interfere with a developer working in normal iteration mode by constantly flooding them with information. But in endgame mode, when developers react to unplanned incoming work, such as bug reports or requests for code reviews, notifications and additional information about relevant events greatly supports their tasks. In this case, Ensemble-like interventions become helpful, whereas obtaining information through tools such as Codebook introduces extra steps to obtain similarly useful information.

Guideline #2: *Communication support systems should assist in non-implementation tasks such as code reviews and risk assessment.* There are many systems that support typical development tasks such as debugging and re-factoring. We found that those tasks have little influence on the information-seeking behaviour of developers. But reviewing or assessing the risk of a changeset has a profound impact on information-seeking behaviour. When developers review or assess the risk of a changeset, they need access to a considerable amount of low level information concerning the changeset, such as information about the motivation behind the changeset and about the code components affected by it. Thus, providing information to developers while they perform tasks such as code reviews or risk assessments supports their current information-seeking behaviour.

Guideline #3: *Communication support systems should account for business goals.* Software teams need to triage and focus the team's efforts on the work items with the highest priority. Some teams record such prioritization under the priority or severity fields of their work items, but these fields tend to have an emphasis on the technical challenge or stability of the product, not on the extent to which the fulfillment of the work item brings the organization closer to satisfy its business goals. Systems supporting the information-seeking behaviour of developers should be able to hide, upon request, the information related to tasks that are not mission critical to the organization, helping the team focus its attention on the most relevant problems.

Guideline #4: *Communication support systems should account for perceived knowledge of other developers.* Several factors that trigger developers to seek information about a changeset are not related to the code in the changeset itself, but to its author. The author's experience level, as well as the quality of her previously delivered work, has a great influence on whether developers will seek information about their most recently submitted work. We suggest to provide a filtering mechanism that accounts for parameters such as author experience and performance.

5.2 Explaining Failure Prediction

Our findings have implications to the current insights in software failure prediction. The literature on failure prediction has identified factors and developed models that cause the software product to be more or less susceptible to failures. These models are often limited in providing explanatory power as to why those particular measures are good failure predictors. Our findings provide possible explanations on a number of powerful failure predictors.

We found that shared work experiences substantially increase the chance that developers will request information from each other. This implies that developers who have never worked together (an occurrence that is more common among members of different teams) are less likely to request information from each other. Lack of familiarity, as an inhibitor of information-seeking behaviour, thus helps explain why organizational distance is a strong failure predictor [21].

Similarly, we found that distance has an insignificant influence on whether they request information about a changeset. This helps explain why Bird *et al.* [4] could not find any relation between the physical distance among developers working on the same binary and the quality of the corresponding binary: developers have adapted to textual, electronic communication media, no matter whether their interlocutors are nearby or halfway around the world.

We observed that the amount of changeset-related information seeking spikes during the endgame of an iteration (see Figure 1). This cyclical repetition of bursts in communication might be linked to the pattern observed by Kim et al. [16] in relation to failures at changeset level. We hypothesize that these bursts could coincide with the development mode the team is in, so that incorporating knowledge about development modes might improve failure prediction models further.

6. THREATS TO VALIDITY

Our survey asked the developers to rank a pre-defined list of items in terms of the influence they may have on their information-seeking behaviour. In doing so, we attempted to reach a compromise between the survey length and survey response rate. This bias poses a threat to our findings due to the possibility that we were missing important items. We mitigated it by developing the survey iteratively by piloting and discussing it with one of the development teams to identify the most important items, and by relying on our other two sources of data to triangulate our findings.

We had the chance to interview ten developers, which represent a fraction of the development team at large. These ten developers were all located at the same site. As a result of this, our interview data could be biased and unrepresentative of the RTC team at large. However, we are confident that this threat is minor, due to the mix of developers we interviewed, including novices, senior developers, and team members that had been part of the group since its beginning. Furthermore, triangulation with our observations and survey responses increases our confidence in our findings.

Our data come exclusively from the IBM RTC development team, and the extent to which our results are valid for other companies is not clear without replications. The RTC development team has more than one hundred members, it interacts with a number of customers, and it is in the process of integrating its product with others. These are all forces that shape communication dynamics. However, their interaction patterns seem to be similar to those of other globally distributed teams that need to synchronize work across multiple time zones.

7. CONCLUSION AND FUTURE WORK

Coordinating work among developers is a keystone of successfully shipping a software product [14, 18]. To fully understand how developers coordinate, it is necessary to understand the factors that influence their communication behaviour. Among our findings, we saw that the current phase in the development process and worker experience were two major factors that influenced whether developers communicate about a change. Tools providing information and research on coordination in software engineering should consider and incorporate our list of factors in order to help developers interact more effectively.

Our long-term goal is to implement a system that recommends key developers to contact about problematic changesets. Looking back at the findings of this study, we can make two conclusions on this topic: (1) there is value in implementing such a recommender system, since developers monitor and assess carefully the information related to changesets in their projects, especially to the extent to which it relates to software quality, and (2) the information-seeking behaviour of developers is affected by more than just the technical nature of a changeset. Individual and process-based information consistently informs their decisions to seek information about a changeset or not.

8. ACKNOWLEDGMENTS

We thank the members of the RTC team for their invaluable participation in our study. We also thank Julius Davies and Irwin Kwan for their comments and feedback on an earlier version of this paper.

9. REFERENCES

[1] J. Aranda and G. Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, May 2009.

[2] A. Begel, Y. P. Khoo, and T. Zimmermann. Codebook: discovering and exploiting relationships in software repositories. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 125–134, New York, NY, USA, May 2010. ACM.

[3] A. Begel and B. Simon. Struggles of new college graduates in their first software development job. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education*, SIGCSE '08, pages 226–230, New York, NY, USA, 2008. ACM.

[4] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy. Does distributed development affect software quality? an empirical case study of windows vista. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 78–88, Washington, DC, USA, 2009. IEEE Computer Society.

[5] M. Cataldo, J. D. Herbsleb, and K. M. Carley. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM '08, pages 2–11, New York, NY, USA, 2008. ACM.

[6] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, CSCW '06, pages 353–362, New York, NY, USA, 2006. ACM.

[7] M. E. Conway. How do Comittees Invent? *Datamation*, 14(4):28–31, 1968.

[8] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Commun. ACM*, 31:1268–1287, November 1988.

[9] 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 *Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*, SIGSOFT '04/FSE-12, pages 221–230, New York, NY, USA, 2004. ACM.

[10] 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

Proceedings of the 2004 ACM conference on Computer supported cooperative work, CSCW '04, pages 63–71, New York, NY, USA, 2004. ACM.

[11] C. R. B. de Souza and D. F. Redmiles. The awareness network, to whom should i display my actions? and, whose actions should i monitor? *IEEE Transactions on Software Engineering*, 99(PrePrints), 2011.

[12] R. Frost. Jazz and the eclipse way of collaboration. *IEEE Softw.*, 24:114–117, November 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 *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, CSCW '98, pages 275–284, New York, NY, USA, 1998. ACM.

[14] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter. An empirical study of global software development: distance and speed. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 81–90, Washington, DC, USA, 2001. IEEE Computer Society.

[15] P. Hinds and C. McGrath. Structures that work: social structure, work structure and coordination ease in geographically distributed teams. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, CSCW '06, pages 343–352, New York, NY, USA, 2006. ACM.

[16] S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller. Predicting faults from cached history. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pages 489–498, Washington, DC, USA, 2007. IEEE Computer Society.

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

[18] R. E. Kraut and L. A. Streeter. Coordination in software development. *Commun. ACM*, 38:69–81, March 1995.

[19] W. G. Lutters and C. Seaman. The value of war stories in debunking the myths of documentation in software maintenance. *Information and Software Technology*, 49(6):576–587, 2007.

[20] G. C. Murphy and E. Murphy-Hill. What is trust in a recommender for software development? In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, RSSE '10, pages 57–58, New York, NY, USA, 2010. ACM.

[21] N. Nagappan, B. Murphy, and V. Basili. The influence of organizational structure on software quality: an empirical case study. In *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pages 521–530, New York, NY, USA, 2008. ACM.

[22] K. Nakakoji, Y. Ye, and Y. Yamamoto. Supporting Expertise Communication in Developer-Centered Collaborative Software Development Environments. In *: Finkelstein, A., van der Hoek, A., Mistrik, I., Whitehead, J. (eds.) Collaborative Software Engineering*, 2010.

[23] A. Schröter, I. Kwan, L. D. Panjer, and D. Damian. Chat to succeed. In *Proceedings of the 2008 international workshop on Recommendation systems for software engineering*, RSSE '08, pages 43–44, New York, NY, USA, 2008. ACM.

[24] T. Wolf, T. Nguyen, and D. Damian. Does distance still matter? *Software Process*, 13:493–510, November 2008.

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

[26] P. F. Xiang, A. T. T. Ying, P. Cheng, Y. B. Dang, K. Ehrlich, M. E. Helander, P. M. Matchen, A. Empere, P. L. Tarr, C. Williams, and S. X. Yang. Ensemble: a recommendation tool for promoting communication in software teams. In *Proceedings of the 2008 international workshop on Recommendation systems for software engineering*, RSSE '08, pages 2:1–2:1, New York, NY, USA, 2008. ACM.