Accountability in Brazilian Governmental Software Project: How Chat Technology enables Social Translucence in Bug Report Activities

Nelson Tenório
University of Copenhagen, Copenhagen, Denmark & Cesumar University Center, Maringá, Paraná, Brazil
nelson.tenorio@unicesumar.edu.br

Danieli Pinto
Cesumar University Center, Maringá, Paraná, Brazil
danicne@gmail.com

Pernille Bjørn
University of Copenhagen, Copenhagen, Denmark
pernille.bjorn@di.ku.dk

Abstract. Fixing software bug is part of the daily work routine in software engineering which requires collaboration and thus has been explored as a core CSCW domain, since the early inception of the research field. In this paper, we explore the use of chat technology in software engineering by analyzing the coordination between client and vendor in a large government software project in Brazil (Gov-IT). We collected our empirical material through face-to-face and online interviews, site and chat forums observations. Looking closely at the bug fixing activities within Gov-IT, we find that the client and the vendor use chat technology to coordinate their cooperative work by enabling the participants to monitor the availability of developers and the urgency of detecting bugs synchronously. This way, the chat technology made it possible for the client to report bugs and developers to resolve bugs in a timely manner. Moreover, the chat technology enabled the participants to request and share artefacts synchronously,

making it possible to analyze and understand the contextual nature surrounding bugs faster than using the bug tracking system. Finally, the chat technology enabled participants in enacting commitment and interdependence across vendor and client, creating cooperative situations of mutual dependence. Our results suggest that we, as CSCW designers, must rethink the design of bug tracking systems and find new ways to re-configure systems, so they support the coordinative practices involved in detecting, analyzing, and resolving critical and severe software bugs synchronously.

Keywords: Coordination, Bug tracking, Chat technology

1 Introduction

In 1947, seventy years ago, Grace Hopper was testing the Mark II Aiken Relay Calculator at Harvard University, where she found an actual moth stuck between points at Relay #70, Panel F. This episode was the first instance, where a machine was 'debugged', and the action of removing the moth from the rely was the first 'bug fixing' activity. While the practices of software development have changed dramatically since 1947, bug fixing is still today an important part of software engineering (Menendez-Blanco et al. 2018). The work of identifying, coordinating, and resolving defects in the software's source code remains a routine collaborative endeavor between software developers.

Detecting, identifying, and resolving bugs in software code is a core activity in software development. The work of identifying and resolving any undesirable or unexpected behavior in the programming code is crucial since defects affect the quality of the software in important ways (Tan et al. 2014). Bug fixing is a rotating task in which involves all the coordinative activities (Cataldo et al. 2006; Šmite et al. 2017) where software developers write and revise source code with the aim of mitigating detects (Zhao et al. 2017). It is critically important to identify and solve software detects in due time, and if software bugs are not resolved appropriately, they generate continuous negative effects regarding software (Akbarinasaji et al. 2018). Large number of bugs within certain systems indicate poor software quality, and failures caused by these bugs are both costly and risky and can potentially harm the user's overall experience (Saha, Khurshid, and Perry 2015). Thus, bug fixing is one of the most significant and indispensable processes in software development and maintenance (Mi and Keung 2016). Bug fixing, as an important collaborative activity, has been explored within prior CSCW research. Early research showed that important coordination in software work includes reporting and classifying identified bugs (Schmidt and Simone 1996). This mean that software developers must find ways to provide necessary and relevant information required to solve the task to enhancing software quality (Zhao et al. 2017) and that they need tools and technologies for this endeavor.

The collaborative nature of software development has resulted in developing tools and integrated development environments (IDEs), which includes technology support for coordination through bug tracking systems, i.e., GitHub, JIRA or Bugzilla (Anvik 2006). Bug tracking systems have demonstrated their worth – especially in large open source projects by supporting the social network and interactions between software developers geographically distributed (Tsay, Dabbish, and Herbsleb 2014). Since Schmidt and Simone (1996) study of the coordinative mechanism back in the early 90s, chat technology has gained ground as part of the core business technology in the organizations (Greif and Millen 2003) supporting software development activities (Hupfer et al. 2004). Furthermore, we have been witnessing how social networking activities from the open source projects (Dabbish et al. 2012) have moved important work in supporting collaboration within the organizations. Thus, the question remains as how to fully comprehend the coordinative practices revolving around bug fixing within geographically distributed organizational software vendor today. Interestingly, we discovered that software developers in Brazil were beginning to use chat technology not only to support internal collaboration across software developers, but also to support the collaboration between software developers and their clients. We got curious on the role which chat technology took in mediating the collaboration and in particular in the context of coordinating bug fixing activities. Therefore, we ask in this paper the following research question: *How does the vendor and the client use chat technologies when coordinating bug fixing activities in a distributed software development project?*

To explore this research question, we initiated a five-month study of the coordinative activities involving bug fixing in a Brazilian governmental distributed software project. We wanted to understand the practices by which software developers coordinated their bug fixing activities including who were involved, and, in particular, what kind of technologies did they use. Moreover, we found that the majority of the software development activities revolved around bug fixing when the project reached the design-after-design stage (Bjögvinsson, Ehn, and Hillgren 2012). Furthermore, we discovered that while the project had access to multiple advanced software development environments supporting the coordination of bugs, software developers chose to introduce and use yet a new technology for their coordination – namely chat technology. Chat technology was used both within the software development team and across the team and the client. We identified three factors, which made the Brazilian software vendor and their client to use chat technology. First, chat technology allows the client to observe the *availability* of software developers (i.e., who is online and able to help immediately), enabling them to swiftly resolve identified bugs in a *timely manner*. Second, chat technology allows both vendor and client to synchronously text (Handel and Herbsleb 2002) to request and share artefacts (i.e., part of codes, screenshots, etc.), which enables developers to reproduce the bugs and fix them *in*

due time. Third, chat technology allows participants (client, software developers, and project managers) to synchronously verify the status of a bug supporting collaboration by creating *mutual dependence*. Together, those three factors facilitated social translucence (Bjørn and Ngwenyama 2009; Erickson and Kellogg 2000) in the bug fixing activities. Chat technology made people, activities, and artefacts visible and available for others to monitor and act accordingly. Chat technology introduces accountability, i.e., 'I know, that you know, that I know', (de Souza and Redmiles 2011) into the collaboration between the vendor and client resolving identified bugs *in due time*. Thus, when 'our everyday social and communication skills are far from sufficient' (Schmidt and Simone 1996), chat technology provides a platform for social interaction across geographical distance (Boden, Nett, and Wulf 2008), in important ways complementing the existing IDE platforms.

The remainder of this paper is structured as follows. First, we revisit existing research on the coordination of software bugs in software development, followed by an introduction of our research method and empirical case. Then we present our results from the empirical analysis, and finally, we discuss the findings and provide our conclusions.

2 CSCW challenges for software Bugs

Studies of the cooperative work involved in the coordination of bug tracking and fixing have been part of CSCW research since the inception of the field (Schmidt and Simone 1996). Software development as a core CSCW domain concerns examinations of the characteristics emerging in the collaborative practice between multiple software developers and related other professional groups when creating software projects (Avram et al. 2009). The interdisciplinary teams involved in software development are multiple and include computer scientists, designers, user representatives, testers, team leads, etc. The work of software development is thus a collaborative situation, where the need for interaction with multiple people causes an interdependence in their work (Smith and Boldyreff 1995), thus requiring articulation work and coordination of activities (Schmidt and Bannon 1992). Therefore, CSCW interest in the area of software development is about understanding the complexity of coordinating cooperative activities with the aim of developing various types of technologies and IT systems, which decrease the complexity of coordination and articulation work (Schmidt and Simone 1996).

Bug fixing is a core activity within the software development domain (Guo et al. 2011) and involves decision making and coordination, identifying the problem as well as figuring out how to implement the solution (Espinosa et al. 2007; Halverson et al. 2006). CSCW researchers have studied bugs in software development process establishing that bug fixing is not a trivial activity, but rather, a complex cooperative activity which involves diverse sets of tools (bug

tracking systems and repositories) and multiple people with different roles and backgrounds, e.g., software managers, developers, testers, users, who are required to cooperate to make the software program function. Developing cooperative technologies to support the process of bug fixing is not an easy task. For example, Breu et al. (2010) analyzed 600 bug reports in the repositories of two large open-source projects found that bug tracking systems needed to be improved to better support three aspects of bug fixing, namely question time, response rate, and response time. For each aspect, they identified several patterns which potentially could improve the functionality of the bug tracking tools, if addressed properly. More concretely, they suggested to help support bug fixing differently over time. At the beginning of a bug's life cycle, the coordination requires many details about the bug, including the context for how and where the bug has been discovered. Often used strategies to provide this extra information includes developers to share screenshots, or small videos reproducing the error (Davies and Roper 2014). Inversely, later in a bug's life cycle, the sharing of the context is replaced by the need to facilitate discussing about correction and bug fixing traceability, rather than the contextual information (Zahedi, Shahin, and Ali Babar 2016). Here technological tools which support frequent questions and answers supporting increased interaction between users and developers to avoid unanswered questions. Assigning bugs for particular people to solve is an important aspect of bug fixing, however, in some cases, the assignment of bugs might be problematic and even harmful. Guo et al. (2011) analyzed the bug reassignment process and found five main reasons for reassignments: finding the root cause, determining ownership, poor bug report quality, hard to decide on a proper fix, and workload balancing. They pointed out that the reassignments are not necessarily problematic because it can be helpful to determine the best developer to fix a bug. In this sense, reassignments improve bug fixing coordination, and the bug tracking system should offer functionalities such as metrics to assess and rate reassignments, knowledge database of experts, and sharpened visualizations of reassignment patterns (Guo et al 2011).

An important artefact in bug fixing is the bug report. A bug report is a form, which contains fields to describe and diagnose software defects aiming to help the developers fix reported bugs (Asaduzzaman et al. 2012; Schmidt and Simone 1996). The bugs are reported by users within a bug tracking system which is the core of the bug fixing process supporting the collaboration between software users and developers (Breu et al. 2010). Besides the system enables the developers to gather and track information about the bugs, which helps developers reproduce and fix them (Davies and Roper 2014).

One of the key challenges for bug fixing in distributed software development projects is the challenge of establishing and maintaining common ground for the task at hand (Bjørn, et al. 2014; Jensen 2014; Jensen and Bjørn 2012; Jensen, Storm, and Bjørn 2011). Common ground is the knowledge that people have in

common and know that they have in common (Clark and Brennan 1991; Olson and Olson 2000). However, knowing the context around a bug from the user who reported it towards the developer who is fixing it is not easy. Software vendors use different communication tools to support their activities (Storey et al. 2017), including email, video conferencing, and other live chat software to keep communication between the software team as well as across team and client (Jan et al. 2016; Pinto, Garcia, and Tenório 2017). Still, challenges persist, especially in situations where the collaborators do not manage to establish and maintain a shared context which means that the risk of miscommunication is high (Bjørn and Ngwenyama 2009).

Research on the use of chat technology in the workspace to aid collaborative work has steadily increased in recent years. According to previous research, chat technology can jeopardizes productivity, since it risk disrupting people in their work, however if notification is directly relevant to an ongoing task chat technology tend to improve the productivity (Czerwinski, Cutrell, and Horvitz 2000). To be successful, new collaborative technologies must be perceived as useful and requires a critical mass of users (Grudin 1994; Herbsleb et al. 2002). Furthermore, when incorporated into programming interfaces, e.g., Eclipse, chat technology is capable of enhancing collaborative work among software developers (Hupfer et al. 2004). Chat technology also can be introduced to students in order to increase their learning with the absence of a tutor. An experiment showed a group of students working collaboratively via chat group was able to engage with the material as if they were collocated (Albin-Clark 2008). Chat technology has also been reported to improve social interaction for large number of users while viewing a live stream together (Miller et al. 2017), and to support distributed teams in building cooperation and trust (Wang and Redmiles 2016) by simplifying the communication and coordinating their activities (Gutwin, Penner, and Schneider 2004).

Based upon prior work, we explore the use of chat technology of the software developers to resolve and coordinate their bug fixing activities in our case. In this work, we address the core research interest for CSCW, namely the cooperative work involved when software developers coordinate and communicate with users on the joint activity of bug fixing, looking particular into the use of chat technology.

3 Method

To investigate how software developers and clients use chat technologies to resolve and coordinate their bug fixing activities in software engineering practices, we got access to do an ethnographically inspired work place study (Randall, Harper, and Rouncefield 2007) in a Brazilian software company. In this work, we look at a Brazilian government software engineering project: GOV-IT.

Our empirical work included a mixture of interviews, observation, and participatory observations in the chat groups. We performed online and face-to-face interviews with the participants following a semi-structured interview protocol. Moreover, we performed observation at the vendor site, also including informal interviews. Finally, one author was added into the five main GOV-IT chat groups, where he did participatory observations over the whole period. Our findings are based on this empirical material, which were recorded, transcribed, analyzed, and finally discussed.

3.1. Gov-IT Project and Chat Groups

The GOV-IT is a long-term project, which was as been initiated in 2011. The purpose of it is to develop and continuously re-configure a public administration system (PUB-SYSTEM) after its implementation. Thus, the software project can be characterized as an ongoing distributed software development. The client for the PUB-SYSTEM is a capital city – municipality - located in the North Brazil: Braavos, while the software vendor is located in a city in Southern Brazil: Volantis. To support the client, the vendor has a team located in Braavos, close to the client of six people, but the majority of the developers are located in Volantis in which work twelve people among developers, team leaders, and project managers. So, in total, the GOV-IT has eighteen people involved in the project.

The PUB-SYSTEM services have approximately 1,200 direct users (i.e., municipality employees) and 250,000 citizens. The first version of the system was deployed in the 2012, and since then additional modules and functionalities have been added to it. The system comprises of four main modules (e.g., tax and fee administration, human resource, financial management, and administrative management), and fifty sub-modules such as accounting, agreements, education administration, public health administration, public patrimony, vehicle fleet, etc. Currently, the PUB-SYSTEM is 75% developed, and since it is already in use new features and functionalities are still continuously being developed. The client and the software vendor established a development-maintenance contract based on Service Level Agreement (SLA), i.e., a formal contract used to guide and structure the coordination. The SLA terms include the differences between bugs and change requests, as well as the time frames by which detected bugs must be resolved. The SLA also includes terms describing financial penalties, where the vendor is required to compensate the client if a certain task has not been solved within the pre-determined timelines and deadlines. In an example, identified bugs in the production environment must be fixed within a two-hour period.

The coordinative platform to organize the work is an official web-based project management and issue tracking tool, Redmine. The client, developers, and project managers register detected issues (bugs and change requests) in Redmine. While bugs are considered defects in software that must be fixed immediately, change

requests include different types of new system functionalities (e.g., a new report or mode to calculate a tax), and improvements of the existing system (e.g., changing the order of input fields on the screen to improve the user interaction), which does not have a strict time frame. Evidently, fixing a bug and implementing a change request can result in negotiations and discussion between the client and the vendor. These negotiations and discussions are accomplished by using chat technology in most cases – but some also include email or phone.

3.2. Data Collection

We collected our empirical material through face-to-face and online interviews, observations, and participant-observation in the chat groups. All material was collected and recorded with the interviewee's permission and informed consent. We started collecting the data by doing a one-hour SKYPE interview with the software vendor, located in Braavos. We followed a semi-structured interview protocol in which the main theme was *processes and communication within distributed software development*. The CEO described and explained the development process and tools used for managing the GOV-IT project. Here, it was evident that the vendor and the client had agreed to use chat technology as a way to coordinate their activities early in the project. We performed five face-to-face interviews at the vendor site focused mainly on the *role* of the chat technology within the GOV-IT project. In these interviews, we re-interviewed the CEO, this time focusing on the chat technology in more details, as well as conducted new interviews with three software developers and one project manager. Three days after the interviews, we performed a four-hour observation at the vendor site. During this observation, we captured empirical data through immediate notes, which were later transcribed. Here the focus was on the software development and communication processes related to the bug fixing activities. Moreover, we had an informal talk with other team members, i.e., two testers, one project manager assistant, and three other developers. Finally, we got permission to participate in the four GOV-IT group chats as an observer, i.e., without interacting with other group participants. Table I summarizes our data sources.

The GOV-IT project was organized through seven SKYPE chat groups, each of them created with a particular focus and specific aim. According to the CEO, we learned that not all chat groups were frequently used, and some of them were about to be abandoned since they had become obsolete. Therefore, the CEO suggested we participate in four concrete chat groups: Namely the tax-group, infra-perf-group, dev-group, and pm-group. Each of these groups had a significant number of daily interactions. From May to November 2017, we observed the four GOV-IT group chats on a daily basis – in total, we spent 560 hours observing and analyzing the interactions. To ensure that we would be notified when important activities took place, we enabled notifications, and each time new messages were

posted in the group chats, we immediately read the messages and analyzed what was going on. All the data were collected for later analyses, such as message content, users, artefacts, etc.

Table I. Data source.

Type of data	*Date*	*Time*
Online interviews		
CEO	27/03/2017	60 min.
Project manager 2 (client site)	05/07/2017	
Face-to-face interviews at vendor site		
CEO	12/05/2017	60 min.
Developer 1	12/05/2017	30 min.
Developer 2		
Developer 3		
Project manager 1		
Observations		
Vendor site	15/05/2017	4 hours
Connected in 4 GOV-IT chat groups (online)	May to November 2017	560 hours

In the beginning, each GOV-IT module had a specific group to tax-group was created aiming to coordinate mostly bug fixing activities regarding the tax and fee administration module of the PUB-SYSTEM. However, as the system functionalities expanded, issues relating to other modules emerged and were also addressed within the tax-group chat. This caused the tax-group chat to emerge as the main communication channel to report bugs for all system's modules (i.e., tax and fee administration, human resource, financial management, and administrative management). Thus, tax-group is the most active group chat and had twelve participants representing both the vendor and the client. During our observation of the group chat, the tax-group chat comprised of 2,138 interactions, with an average of 10.28 messages per day (detailed in Table II), of which 15% of those messages are related to bug fixing, and 75% are related to the new request.

The second most active group chat was the infra-perf-group chat. This group chat was dedicated to discussing the infrastructure and performance of the PUB-SYSTEM and was concerned with issues around infrastructure and operation. Topics included slow queries, web services which did not work, system access fails, server crash, communication fails, and so on. The infra-perf-group chat had twelve participants representing the vendor and the client. We observed 932 messages in the infra-perf-group chat with 4.44 daily messages on average, as shown in Table II.

Finally, the third and fourth most active group chat were the pm-group and the dev-group chats. Both of those groups chats had only vendor participants. The om-group chat was dedicated to the issues of the project management for the GOV-IT (i.e., deadlines, costs, priorities, and distribution of tasks), while the dev-group chat focused on topics related to technical matters occurred during the

programming activity such as errors of logic, tools configuration, and development environment. The pm-group chat had seven participants located in the vendor site – these included directors and developers – and one participant located in the client site. The dev-group chat included ten developers and one project manager all located at the vendor site. During our observation of these chat groups, we observed 191 messages sent in pm-group with a daily average of 0.91 and 53 in the dev-group chat with a daily average of 0.25 per day (see Table II).

Table II. The chat groups setup.

	Chat groups							
	tax-group		*infra-perf-group*		*pm-group*		*dev-group*	
# of Messages in the period	2,138		932		191		53	
Average per day	10.28		4.44		0.91		0.25	
Participants	*Client*	*Vendor*	*Client*	*Vendor*	*Client*	*Vendor*	*Client*	*Vendor*
Vendor site	N/A	CEO (1) CIO (1) PM (1) Dev (3)	N/A	CEO (1) CIO (1) PM (1) Dev (2) Sup (1)	N/A	CEO (1) CIO (1) PM (1) PMA (1) Dev (3)	N/A	PM (1) Dev (10)
Client site	CIO (1) DT (1) ExL (1)	PM (1) Dev (1) Sup (1)	CIO (1) ExG (1) ExS (1)	PM (1) Dev (1) Sup (1)	N/A	PM (1)	N/A	N/A
Total by site	3	9	3	9	0	8	0	11
Total of Participants		12		12		8		11

(CIO) Chief of Information Officer; (Dev) Developer; (DT) Director of Technology; (ExL) Expert in tax laws; (ExG) Expert in IT Governance; (ExS) Expert in Data Security; (Sup) Technical Support; (PM) Project Manager; (PMA) Project Manager Assistant; (N/A) Not Applicable.

3.3. Data Analysis

We analyzed the data iteratively, so that early insights could structure later activities, e.g., online interviews on chat technology in software engineering led us to perform face-to-face interviews on the use of chat technology for bug fixing, which let us observe chat groups, etc. Thus, we analyzed the interview transcriptions and coded the material by highlighting important incidents in the material (i.e., actions that caught our attention) by following themes such as bug fixing, infrastructure issue, development process, questions, and user support. At the same time, we analyzed interaction in the chat groups across all the chat groups. All those data were gathered and analyzed in Atlas TI. Our analysis revealed that the majority of the interaction in the chat groups were related in different ways to bug fixing – and thus this became our main focus point. We

define bug fixing situations in the data material as 'incidents' or situations where interaction, coordination, and communication were accomplished in the group chats to resolve detected bugs. In all these situations, we investigated the ways in which participants used tools, techniques, and other tactics, to accomplish their work and resolve bugs in a timely manner.

4 Results: Bug fixing in Gov-IT project

4.1. Introducing chat technology to coordinate software bugs

In November 2012, a severe bug stopped the operation of an important PUB-SYSTEM module. The citizens depend upon the PUB-SYSTEM to pay their taxes. Thus the system breakdown seriously affected important work of the municipality. At the beginning of the GOV-IT project, both vendor and client had agreed to adopt REDMINE as a tool to coordinate all the required aspects of change requests and bug fixing. Following the established protocol, the client registered the bug onto REDMINE. However, the REDMINE system failed and did not notify the vendor about the severe bug in due time. Instead, the vendor was only informed about both the severe problem and the system breakdown three hours after the bug had occurred. Instead, the vendor has some hours dedicated to fixing bugs found in the production environment, accordantly to the SLA critical bugs should be fixed within two hours, and the vendor was forced to pay a financial penalty to the client since they had not learned about the bug situation before it was too late. After this incident, the vendor proposed and convinced the client that they were to use chat technology into their collaboration. The purpose of introducing the chat technology was in order to coordinate all the GOV-IT activities related to urgent bugs and to avoid future financial penalties.

> 'We have established an SLA, which dictate that we must fix severe bugs within 2 hours, or else we must pay financial penalty to the client. The system [PUB-SYSTEM] stopped and we had a hard time at the beginning of the project. In November of 2012, the REDMINE stopped working suddenly, and none of us were notified about that failure in the system. When we learned about the issue, it was too late. […] So, we paid the [financial] penalty to the client, and I did an emergency meeting with my whole team, and after that another meeting with the client. Here we decided to introduce SKYPE [chat technology] and use this to coordinate our bug fixing activities within the group chats. We still use REDMINE as a repository to gather information about change request, but not as a tool for coordinating bug fixing anymore. So, we institutionalized SKYPE and, since then, we have not paid penalties to the clients anymore.' (23 March 2017, Vendor CEO, Online Interview, translated from Portuguese).

The vendor created four different chat groups based upon what was experienced. Each of these with the purpose of coordinating specific types of bugs and ensuring that serious bugs were resolved immediately. The vendor and client

agreed that team members from both sites should stay online during working hours, to ensure that any kind of issue reported in the chat technology would be addressed appropriately. To further strengthen the coordination of the detected bugs, the vendor organized the group chats into topic-vise sub-groups as an approach to segregate the reported bugs and streamline the discussions based on the relevance of participants. The structure chosen is characterized as a top-down topic one, where each module of the PUB-SYSTEM has its own group chat. For example, tax-group discussed subjects about tax and fee administration module; hr-group discussed topics regarding human resource module; fin-group discussed topics related to the financial management module; and finally, adm-group discussed topics concerning administrative management module. However, as the system expanded, and additional functionalities were added, all related bug discussions were migrated to the tax-group chat.

> 'The system [PUB-SYSTEM] grew up considerably, and the client started reporting bugs to tax-group. I think it happened because this module [tax and fee administration module] is the core of the system.' (12 May 2017, Project Manager at the vendor site, Interview, translated from Portuguese).

This merge in topics emerged because the module of tax and fee administration was the most important one of the PUB-SYSTEM, and thus where most issues were reported. As reported by a developer during our interviews at the vendor site, the tax-group chat became the main group chat for the coordination of the GOV-IT project, while the other group chats were being deactivated since they were obsolete.

> 'The client started reporting bugs in the tax-group, and we answered that, maybe because the tax module [tax and fee administration module] is the main module of the system. Actually, we are deactivating the other groups […] They have become obsolete' (12 May 2017, Developer at the vendor site, Interview, translated from Portuguese).

The tax-group chat includes multiple participants from the project such as the client, project managers, and developers. In more concrete terms, the group is used to coordinate the collaboration and bug fixing activities sending out notifications to the whole group of participants such as asking people to check a particular bug already reported in REDMINE or even presenting bug details from the messages shared in the group chat. Moreover, the vendor created three private group chats to coordinate GOV-IT internal activities in which only participants from the vendor team were present. One private group chat was dedicated for the developers addressing topics related to development processes, practices, and programming environment, errors of logic, and tools configuration. The other private group chat was the project manager group chat which was dedicated to addressing and discussing topics related to the management of the GOV-IT such as time appointment, deadlines, costs, priorities, and distribution of tasks. The third private group chat was dedicated to coordinating the deployment and building the

tasks of the PUB-SYSTEM, that was used to notify developers and project managers when the new PUB-SYSTEM was released. This way, all private group chats have become important tools for organizing and coordinating the vendor team, who were located at different geographical sites. Thus, the vendor and the client formalized the chat technology as an institutionalized communication tool to coordinate their bug fixing activities within the GOV-IT project.

Zooming in on the ways the vendor and the client used chat technology to coordinate bugs, it was clear that the bugs discussed could be categorized into two categories: Change requests and bugs. While bugs are issues, which needs to be solved depending on the urgency and severity. Change requests are new features, which the users would like in future releases. Below is an example of a change request #8052:

> 'Request change #8052 - Tax module – Generates parcel report - opened.' (07 November 2017, Client in the tax-group, 18:04, translated from Portuguese)

The change request #8052 is a conventional example from GOV-IT of an event, where the client informs the vendor about a new change request, in this case, called *report of payments parceled*. In this example, the client decided to ask for a change request and added this new request to REDMINE and then changed its status to open. The vendor team see the notification in the group chat and immediately begins to work on the task in order to resolve the issue. In this case, the developer solved the change request within 4 minutes from being reported, the vendor respond in the group chat that the issue has been resolved and is ready to get validated by the client. One hour later, the client sends a message to the group chat saying that the change request has been validated.

> 'Request change #8052 - Tax module – Generates parcel report – ready to validate.' (07 November 2017, Client in the tax-group, 18:41, translated from Portuguese)

> 'Request change #8052 - Tax module – Generates parcel report - validated.' (07 November 2017, Client in the tax-group, 19:45, translated from Portuguese)

When the participants in the PUB-SYSTEM uses the chat technology in the above examples, its main use is related to coordination. It is important to notice that all these change requests and bugs are all registered upon REDMINE. Thus, it would have been possible to do all this coordination within the official system. Still, both vendor and client choose to spend the extra effort and send notifications of the status change within the group chats. Consequentially, all the coordination between them was done outside the official bug handling system. As seen in the above examples, the chat technology has become an essential tool for improving the ways in which the team members organized and coordinated severe bugs, which needed to be solved right away. In this sense, the chat technology has become a way for them to ensure redundancy of the critical work process of

severe bugs in the production environment. So, the chat technology has become a crucial tool for the vendor to ensure they will not need to pay the financial penalty. They did not rely on REDMINE, due to the previous system failure. Furthermore, it is important to highlight, that it was not because the status change information was not recorded in REDMINE, it was. However, this status information was not used in the coordination process, but mainly as a repository of the information surrounding the bugs.

4.2. Software developer availability

Clearly, the chat technology is an important tool for the coordination of bug reports, however as it turned out it was mostly used in certain situations of bugs – namely when the urgency in fixing the bugs was critical. The GOV-IT group chat was created intending to coordinate the bug fixing activities involved in the PUB-SYSTEM operation, mainly the tax and fee module since it was in use and was critical for the client. This module is the core of the PUB-SYSTEM, representing a critical activity for the government, and thus potential bugs identified need to be solved immediately. As in all the other modules, each newly identified bug had to be recorded on REDMINE for the PUB-SYSTEM module. After the bug has been recorded the developers evaluate whether the bug reported is a bug or a change request, as well as estimate the time required to solve the issue.

However, analyzing the group chat discussions, we found three situations, where the bugs were not recorded on REDMINE, but only coordinated via group chat. Thus, the complete activities around resolving the bug were performed outside REDMINE. An example of this workaround happened with *single invoice generating* functionality. The *single invoice generating* is an essential feature of the PUB-SYSTEM, which is a critical system for the municipality. A *single invoice* enables citizens to do business without opening up actual shops and still remain as a legal entity. The policy is used particularly to support new start-up companies, allowing individuals who starts their own business to do their business as well as pay their taxes. In Braavos city, the PUB-SYSTEM generates around 1,000 single invoices per day, every time a tax payment is made. This action is documented in the system database, which then creates a pdf file, to be printed and shared with that citizen.

Since it is a critical system, all bugs related to *single invoice* must be solved quickly. During our observation, the client reported a PUB-SYSTEM severe bug. The problem was that the PUB-SYSTEM got one of its screens frozen, when a *single invoice* was being generated, which basically meant that the user was unable to operate the system. The user tried several times to generate a *single invoice*, but the PUB-SYSTEM would still have its screen frozen continuously froze its screen. This was a severe problem, and solving that issue was an urgent matter since the functionality of *single invoice generating* was part of the production

environment. Concretely, citizens all around the municipality were basically waiting to pay their tax and get their single invoices, while developers needed to fix the defect. However, instead of reporting the bug on REDMINE, the client chose only to contact the vendor via tax-group chat:

> 'Hey guys, after the 'build' at 1h30 p.m. bug is occurring when we register a new [single] invoice and try to save it. The system's screen is freezing. It was working fine earlier! Could you guys check what's going on? All clients [referring to citizens that were paying the tax at that moment] are impatient here!' (05 September 2017, Client in the tax-group, 14:22, translated from Portuguese)

What we see in the quote above is the client reporting on the urgency of the problem, explaining the situation of citizens trying to pay their taxes while highlighting its emergency by referring to the impatient citizens. It is interesting that the urgency of the situation made the team totally skip the formal protocol to report the bug in REDMINE and instead go directly to the tax-group chat. In this situation, the tax-group chat allowed the client to contact Braavos quickly because through of the chat technology, the client, were able to monitor who from the vendor group were online at this time, and thus know about their *availability* to attend to the urgent bug. Thus, the chat group allows the vendor members to fix the bug in a timely manner. One minute later, one developer responds to the client in the tax-group chat, which is very fast compared to if the client had spent time to register the bug in the REDMINE before reporting the issue in the chat group it would had taken longer time. Although the chat group offers an advantage in terms of transparency to reporting severe bugs in emergency cases and be promptly answered, the pitfall of not registering the bug accordantly to policy in REDMINE is the persistence and history of events. The history of the bug reported and later fixed can easily be lost of only reported with the chat technology, and the vendor can face some difficulty to solve similar bugs with the same characteristics since the chat tools does not offer functionalities to organize the messages sent in the groups. Instead, over time – all messages will be organized accordantly to chronology and not easily searchable.

4.3 Sharing artefacts to fix bugs *in due time*

We also observed another interesting use of the chat technology related to bugs, namely how the participants used the tax-group chat to produce the contextual information around the bug for identifying a way to solve the problem. When a bug was reported, the developers need detailed contextual information in order to resolve the problem. Let us look at an example. Antonio, one developer, collocated at the vendor site, request additional information about the severe bug related to the *single invoice* problem. Thus, he asks in the chat group, whether he

can ‘see’ an example of *single invoice* data, which causes the screen to freeze. He needed this information to figure out how to solve it, by understanding the reasons behind the issue.

> ‘Can you send me one example of the bug, please?’ (05 September 2017, Antonio in the tax-group, 14:23, translated from Portuguese)

Antonio could not reproduce the bug, so he required additional information from the client concerning the context of the bug. Therefore, Antonio asked the client to share the record saved in the PUB-SYSTEM database. However, Antonio had not realized that the client could not send him the invoice record once the invoice information where not being saved in the database. As the client explain to him:

> ‘I can’t send you one example because it is not saving [referring to PUB-SYSTEM database]. I’m waiting here [save the single invoice in the database] more than 5 minutes, and it’s not working.’ (05 September 2017, Client in the tax-group, 14:23, translated from Portuguese)

The whole client computer was frozen while the invoice was not recorded in the database. Thus, nothing could be done to send that information. Therefore, they begin to discuss how the developer can get more contextual information. At first, Antonio does not really comprehend the issues as why he cannot get a copy, but through their discussions, he begins to realize the problem.

> ‘Ok. So, send me one [example, i.e., the record] saved in the database.’ (05 September 2017, Antonio in the tax-group, 14:24)

> ‘How can I send it to you if it even not being saved???’ (05 September 2017, Client in the tax-group, 14:24)

Clearly, none of the obvious ways to understand the problem and for the developer to reproduce the bug was applicable. To solve the problem the client decides to take a picture of the screen displaying the problematical feature and then share it in tax-group promptly. This client decision allowed the developer to gain more contextual information and thus gain insights into the critical bug:

> ‘Antonio, I’m sending a picture! It shows the invoice data informed by me. Thus, you can simulate the behavior there [at vendor site]. I did it on my computer, but I tried on another computer, and happened the same problem.’ (05 September 2017, Client in the tax-group, 14:25)

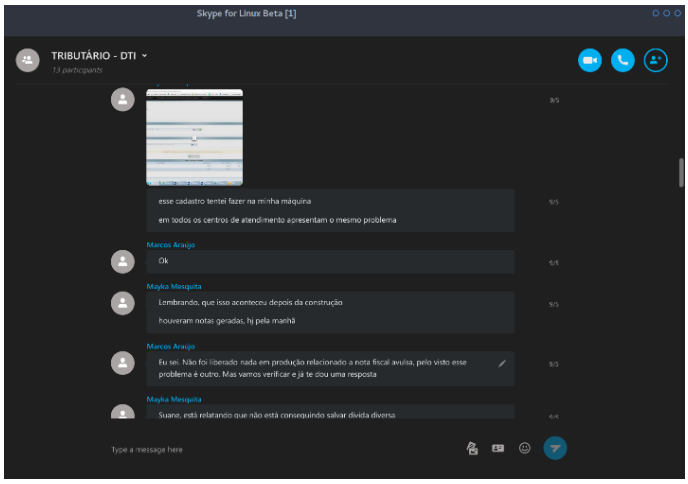

Figure 1. Client sharing a picture to help bug fixing.

About thirty minutes after the bug had been detected and reported in the chat group, Antonio warned the client that he had been able to reproduce the bug, and he had identified that it was an SQL code which caused the 'freezing' of the system. At that moment, another developer, Carlos, decides to collaborate and asks the client to send him more information regarding the bug. Especially he tries to uncover how the input-data have affected the operation of the PUB-SYSTEM. Carlos request that the client provides him with the 'database status' (a set of database server resources, e.g., memory, disk, and CPU, necessary for performing the database's tasks). A few minutes later, the client, then helped by a database administrator collocated at the client site, shares an SQL code in tax-group chat:

> 'Hi, could you ask your DBA [database administration at client site] send us the "database status"?' (05 September 2017, Carlos in the tax-group, 15:00)

> 'Yes, just a moment, please. I'm checking this with him.' (05 September 2017, Client in the tax-group, 15:00)

> 'See what the DBA informed to me [client pasted following dialog]: "This SQL instruction is using all database server resources: [Complete SQL instruction]".' (05 September 2017, Client in the tax-group, 15:03)

Reproduce a bug is an essential software developer strategy to understand and fix it. While the bug systems offer artefact-sharing and bug-reporting functionalities, this is often not enough to reproduce the contextual information required to solve an urgent bug. When it comes to urgent bugs, the importance of swiftly interaction between the vendor and the client is essential, if they are to fix the bug in due time. There the synchronous functionality of the chat technology turned out to be of an essence. In the GOV-IT project, the possibility of

synchronous collaboration offered by chat technology was a substantial advantage to solve severe bugs promptly, compared with the traditional bug fixing protocol established at the beginning of the project. What makes this case interesting is that the client makes use of the synchronization feature advantage of the chat technology to share different sorts of artefacts (e.g., picture and code) address to aid the developers to reproduce the bug and then solve the problem. Moreover, we observed several different urgent bug reports in the production environment. In each case, we saw how artefacts were shared in tax-group chat to assist the developers, and even the project managers, to reproduce, understand, and synchronously fix urgent bugs.

4.4 Mutual dependence

We observed several cases in which tax-group chat made it possible for the distributed team to collaborate across sites fixing severe bugs promptly. In those cases, both vendor and client enacted *mutual dependence* between the developers involved in resolving the bug and the client who had detected the bug. The mutual dependency turned out to be an important feature to ensure the collaboration had successful results. To illustrate the importance of *mutual dependence*, we continue exploring the *single invoice* example as the severe bug in the production environment. During our observation in tax-chat group, we notice that the client suggested that the PUB-SYSTEM build from the day before might had caused the problem. However, Antonio disagrees with the client arguing that 'yesterday' system build activity did not impact on the *single invoice* functionality. In their attempt to fix the issue, this turned out to be an important discussion.

> 'I'd like to reinforce that this problem began after the deployment of [new features in the production environment] because this morning it was working normally.' (05 September 2017, Client in the tax-group, 14:24, translated from Portuguese)

> 'Ok, I got it, but we did not deploy anything related to invoice. I think it isn't related to the new system deploying. I'll check this out and return to you.' (05 September 2017, Antonio in the tax-group, 14:25, translated from Portuguese)

What we see here is an example of how the client and the developer are mutually dependent upon each other in the joined task of identifying the reason for the detected bug. We see how they exchange ideas of how the bug was created and when it was created. Following the above discussion, the client begins to report additional problems related to a different set of PUB-SYSTEM functionality trying to prove his assumption that the previous deployment of the system trigged the bug. The quotes below demonstrate a tension which begin to emerge due to the rapid interaction and opportunity for the client to continue interacting with the

developers – even though the developers would prefer not to be interrupted and instead have time to work on the problem, as established in SLA contract.

> 'Another user is not able to save the public debts [another system functionality]. It's worrying, what's happening??? Peter [project manager] and Carlos [another developer] could you say something about it?' (05 September 2017, Client in the tax-group, 14:27, translated from Portuguese)

> 'I already told you that we are figuring it out. I'll inform you, ok?' (05 September 2017, Antonio in the tax-group, 14:28, translated from Portuguese)

> 'Sorry, I'm just reporting another [case of the same] bug.' (05 September 2017, Client in the tax-group, 14:27, translated from Portuguese)

> 'We already know that.' (05 September 2017, Antonio in the tax-group, 14:29, translated from Portuguese)

> 'OK!' (05 September 2017, Client in the tax-group, 14:29, translated from Portuguese)

The client is dependent upon the developer to resolve the bug and is unable to continue their work before the developers have solved the problem. In this way, the client is dependent upon the developers to solve the problem, while the developers are dependent upon the client to learn about the background of the bug. In this way, the client is dependent upon the developers to solve the problem, while the developers are dependent upon the client to learn about the background of the bug. In this way, the above demonstrates the *mutual dependence* provided by chatting technology when, firstly, the client generates an interdependence on actions between two developers once he points out his bug-cause assumptions trying to help them; and secondly, he seeks answers from the developers, and he involves the project manager, regarding the status of the bug fixing. The bug tracking systems systematize bug information allowing both client and vendor to know details about the bug such as; who is fixing the bug and what is the bug fixing status. On the other hand, those systems are highly dependent on the bug report quality informed by the users, and it is the primary barrier to be overcome in those kind systems. However, while chat technology allows *mutual dependence* once the client can suggest and follows bug fixing status synchronously, it increases the stress between the vendor and the client, such we observed in this case. Therefore, the chat technology affords the interaction by making it possible for others to monitor the interaction between others and act accordingly.

5 Discussion

New technologies have entered software engineering since Schmidt and Simone (1996) wrote their paper on a coordinative mechanism more than 20 years ago. Bug tracking systems are designed to coordinate all the work of identifying, reporting, and fixing bugs during software development processes. The basic design of bug tracking systems is to integrate and reduce the efforts of coordination by supporting the extra work of handling dependencies among tasks and people (Šmite et al. 2017) and thus reduce cost and time – and hopefully achieve improved IT products (Espinosa et al. 2007).

Surprisingly, we found that introducing chat technology to the bug fixing practices, the software developers did not aim to reduce efforts of articulation work, as expected. If that had been the case, the focus would be to reduce the connections and dependencies within the work. Instead what made them introduce chat technology was an interest to increase the social interaction – and thus increasing the required efforts of articulation work with their remote colleagues. So rather than lowering connections and dependencies within the work, supported by the asynchronous nature of the existing bug tracking system, the vendor and client used the chat technology as a vehicle to coordinate their bug fixing activities synchronously. They increased the dependencies across sites (including the client) thus producing closely-coupled work arrangement. The closely-coupled work arrangement was important for them because the GOV-IT system was already in-use, thus users replied upon the stability of the functionalities. Thus, closely-coupled synchronous interaction was required to handle the complexities of the 'design-after-design' nature of the software development work (Bjögvinsson et al. 2012). Despite the participants being geographical dispersed, the chat technology added functionality, which the existing bug tracking system Redmine could not provide. Other research has identified closely-coupled work as supportive for collaboration across geography (Bjørn et al. 2014; Jensen 2014) because it forces frequent interaction despite collaborators being dispersed. Our work suggests that by introducing chat technology they did not reduce efforts of articulation work, but instead increased the dependencies forcing frequent interaction across vendors and clients.

Social interaction in programming activities is salient in open source projects (Dabbish et al. 2012). Here the work is often asynchronous and by default geographical distributed. What makes open source project function, is the ways in which the development platform (e.g., GITHUB) provides ways for the developers to monitor the collective effort of the team, faciliting individuals to know in what ways their current tasks are dependent upon the tasks of others. In this way, the development platform provides transparency in open source work. So how come the existing development platform Redmine not being enough for the client and developers in our case? What was it lacking?

Analyzing our case, it became clear that rather than introducing chat technology as a substitute the face-to-face communication, the SKYPE groups served as a new mechanism of interaction in supporting coordination between the software developers and the client. The chat technology made it possible for the client to monitor the availability of the software developers, who could resolve the detected bugs in a timely manner. In addition, the chat technology also made it possible for the developers to monitor urgent detected bugs in due time ensuring that bugs were resolved following the contractual agreements avoiding financial penalties. In this way, the chat technology introduces accountability to the collaborative relation between the client and the software developers (de Souza and Redmiles 2011). The chat technology added social translucence (Bjørn and Ngwenyama 2009; Erickson and Kellogg 2000) into the IT eco-system comprising REDMINE, email – and the chat technology, which together supported the collaboration in different ways. On the one hand, the chat technology provided accessibility for collaborators to make visible *while* monitoring each other's actions, and thus allowing individuals to act accordantly - and ultimately create accountability. When we look at the use of chat technology in the bug fixing activities, we see how the technology provides the developers the ability to monitor the interaction of others. Because of the nature of the chat technology being technically setup as multiple group chats, each with different sub-topic areas, made it possible for the team members to not only monitor specific individuals – but instead monitor and have access to the complete set of interactions which all developers have with the client. In this way, they are able to continue the work of others explicitly coordinating with fellow developers.

Early work on the use of chat technology in the workplace pointed out how the employees did not find chat useful in their everyday interaction, but instead perceived the chat technology as a superfluous tool not adding anything significant to the coordination (Cataldo et al. 2006; Herbsleb et al. 2002). Our empirical data from the GOV-IT project displays an entirely different insight about chat technology in software development. In our case, the chat technology was perceived by the developers and the client as beneficial and useful - thus adding new functionality to the ecology of systems, which the current development platform was not able to do.

Let us return to our research question: How does the vendor and the client use chat technologies when coordinate bug fixing activities? In our analysis, we identified different usages, which was essential for the choice of introducing chat technology into the collaboration. We found that the chat technology provided the participants new ways of monitoring the availability of developers as well as monitoring the urgency of detected bugs in an accessible and synchronous way supporting coordination. In Brazil, 73% workers from the software industry reported using WHATSAPP and SKYPE as important chat technologies on regular bases and in various different types of work situations (Pinto, Garcia, and Tenório

2017). This suggests that the challenge related to cost/benefit in groupware technology adaption (Grudin 1994) was solved by supporting the different yet dependent needs for synchronous collaboration to coordinate the work (Gutwin et al. 2004). Chat technology used in the GOV-IT project enabled the client to report severe bugs, and developers to fix detected bugs in a *timely manner*. The chat technology allows software developers to interact closely with the remote client supporting both spontaneous and opportunistic communication Herbsleb et al. (2002). Finally, the chat technology functioned as a mean to request and share artefacts synchronously making it possible to resolve severe and urgent bugs *in due time*. A feature, which is important for both the client and the vendor. For the client, chat technology allowed the participants, who are stuck in a breakdown situation working with a detected bug in a 'design-after-design' system (Bjögvinsson et al. 2012) to continue their work with citizens in the municipality accordantly to schedule. For the vendor, the chat technology allowed the participants to learn and know about the bugs *in due time* reducing the risk of financial penalties. Finally, chat technology enabled software developers in demonstrating commitment when coordinating their individual yet *mutual* dependent work activities (Schmidt and Bannon 1992). The mutual dependence emerges in the case, because of the time constraints giving by the contractual agreement stipulating that participants must verify synchronously whether severe bugs have been detected and solved.

The informal nature of the chat technology, combined with the rapid exchange of messages as well as the persistent of the shared interaction and the ability to link in other materials, like the screen shot, made the chat technology the preferred tool for coordination between the vendor and the client for bug fixing. So, while they did have access to a formal coordination mechanism (REDMINE), which was designed to reduce the effort of articulation work, we found that in our case, it was not about reducing articulation work – but instead about providing a different type of cooperative interactions. Our results suggest that we as CSCW designers should consider how to re-design and re-configure bug tracking systems, to not only reduce the effort of articulation work – but also support the coordinative practices involved in detecting, analyzing, and resolving urgent and severe software bugs synchronously. Such designs should take into account how participants are able to report detected bugs in a *timely manner* and request and share artifacts explaining the contextual nature and details important for resolving bugs in due time. By introducing chat technology to the bug fixing activities the software developers were able to facilitate their mutual dependencies involved in coordinating bug fixing activities by providing social translucence to the interaction across geographical sites, software developers, and users. The above contribution expands prior CSCW research by introducing social translucence (Erickson and Kellog 2000) to the practices of software development. Prior CSCW research have demonstrate how awareness is an important feature of the

collaborative work of software developers. For example, Gutwin et al (2004) identified how awareness was an important feature of open-source software development, and De Souza and Redmiles (2011) continues this work arguing for the critical role of awareness network providing information to the developers about 'who should be monitored and to whom their actions should be displayed.' Extending that, our research shows that the bug fixing coordination goes further than monitoring the work of distributed developers. We argue that chat technology not only indicates the availability of the software developers, but also enable developers and users, who are mutual dependent in their work to coordinate bug fixing activities by providing social translucence. Social translucence extends awareness by making activities and practices visible available for others to monitor and act accordantly introducing accountability to the distributed work (Bjørn and Ngwenyama). When, Handel and Herbsleb (2002), discuss the advantages of chat technology in the workspace as a potential communication tool to coordinate tasks in workspace, our data demonstrate how and in what way. While Cataldo et al. (2006) show how awareness is important when designing coordination tools, we argue the importance of designing tools for bug tracking and fixing supporting social translucence, and that chat technology functionality is a viable approach.

Chat technology supported the software developers in interacting with the users concerning bugs identified in the IT system-in-use. Thus, our data shows the benefits in how chat technology enables software developers and users in resolving bugs. However, before we replace interaction concerning the coordination of bugs with chat technology, it is important we also consider the potential problems, which chat technology entails. While the informal language supported the software developers and users in our case, using informal language might also risk causing miscommunication (Handel and Herbsleb 2002). This challenge is based upon the fundamental challenge of creating common ground within geographically distributed teams (Bjørn et al. 2014; Jensen 2014; Jensen and Bjørn 2012; Jensen, Storm, and P. Bjørn 2011). Chat technologies does not automatically allow collaborators to establish common ground, since it depends upon whether the collaborators are able to establish and maintain a shared context including professional and domain specific languages (Bjørn and Ngwenyama 2009). By providing the opportunity for open-ended interaction, chat technology is a malleable technology, which was important for the software developers in our case. However open-ended and malleable technologies also risk an increase of irrelevant interaction and experiences of wasting another people time (Wang and Redmiles 2016). Chat technology enable certain open-ended interactions supporting social translucence in the communication; but chat technology can also constrain work and activities if the technology-in-use patterns are not structured, developed, and agreed upon by the collaborators in explicit or implicit protocols of use.

6 Conclusion

Software development project is immediately collaborative and has always been a core CSCW domain to explore. With the increased use of diverse sets of social computing technologies in the workplace, we are also witnessing how the definition of workplace technologies are changing. CSCW technologies are no longer 'just' about decreasing the efforts of articulation work through coordination, but instead serve as the technological foundation of cooperative practices within the workplace. In our case, we found that chat technology served as a technological platform for mediating social translucence across the vendor and client by allowing to display and monitor activities and creating accountability in the coordination activities.

Despite having access to multiple advanced software development environments supporting the coordination of bug fixing activities, we found that the software developers in Brazil chose to use chat technology for their coordination both within and across organizations. By using the chat technology, the software developers were able to organize timely coordination, ensuring the newly detected bugs were resolved quickly, and that contractual agreements were held reducing the risk of financial penalties. Our work proposes to re-think functionalities in bug tracking systems and include considerations for how to support synchronous interaction in closely-coupled work across distance – ensuring that bugs are fixed *in due time*.

7 Acknowledgments

We would like to thank Cesumar Institute of Science, Technology, and Innovation (*Instituto Cesumar de Ciência, Tecnologia e Inovação* – ICETI), Maringá – Paraná – Brazil; MGA Public Management, Maringá – Paraná – Brazil; Program to Support Private Higher Education – PROSUP of Coordination for the Improvement of Higher Education Personnel – CAPES, Brazil.

8 References

Akbarinasaji, Shirin; Bora Caglayan; and Ayse Bener (2018). Predicting bug-fixing time: A replication study using an open source software project. *Journal of Systems and Software*, vol. 136, no. C, February 2018, pp. 173–186.

Albin-Clark, Adrian (2008). Virtual chat in an enquiry-based team project. *ACM SIGCSE Bulletin*, vol. 40, no. 3, August 2008, pp. 153–157.

Anvik, John (2006). Automating bug report assignment. In K. M. Anderson (ed): *ICSE'06. Proceedings of the 28th International Conference on Software Engineering*, Shanghai, China, May 20–28 2006 . New York: ACM Press, pp. 937–940.

Asaduzzaman, Muhammad; Michael C. Bullock; Chanchal K. Roy; and Kevin A. Schneider (2012). Bug Introducing Changes: A Study with Android. *ICSE'12. Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, Zurich, Switzerland, June 2–3

2012 . Piscataway, New Jersey: IEEE Computer Society, pp. 116–119.

Avram, Gabriela; Liam Bannon; John Bowers; Anne Sheehan; and Daniel K. Sullivan (2009). Bridging, patching and keeping the work flowing: Defect resolution in distributed software development. *Computer Supported Cooperative Work*, vol. 18, no. 5–6, 2009, pp. 477–507.

Bjögvinsson, Erling; Pelle Ehn; and Per-Anders Hillgren (2012). Design Things and Design Thinking: Contemporary Participatory Design Challenges. *Design Issues*, vol. 28, no. 3, December 2012, pp. 101–116.

Bjørn, Pernille; Morten Esbensen; Rasmus Eskild Jensen; and Stina Matthiesen (2014). Does Distance Still Matter? Revisiting the CSCW Fundamentals on Distributed Collaboration. *ACM Transactions on Computer-Human Interaction*, vol. 21, no. 5, November 2014, pp. 1–26.

Bjørn, Pernille; and Ojelanki Ngwenyama (2009). Virtual team collaboration: Building shared meaning, resolving breakdowns and creating translucence. *Information Systems Journal*, vol. 19, no. 3, April 2009, pp. 227–253.

Boden, Alexander; Bernhard Nett; and Volker Wulf (2008). Articulation work in small-scale offshore software development projects. *ICSE'08. Proceedings of the 2008 International workshop on Cooperative and Human Aspects of Software Engineering*, Leipzig, Germany, May 10–18 2008 . New York: ACM Press, pp. 21–24.

Breu, Silvia; Rahul Premraj; Jonathan Sillito; and Thomas Zimmermann (2010). Information needs in bug reports: Improving Cooperation Between Developers and Users. *CSCW'10. Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work*, Savannah, Georgia, USA, February 6–10 2010 . New York: ACM Press, pp. 301–310.

Cataldo, Marcelo; Patrick A. Wagstrom; James D. Herbsleb; and Kathleen M. Carley (2006). Identification of coordination requirements: Implications for the Design of Collaboration and Awareness Tools. *CSCW'06. Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, Banff, Alberta, Canada, November 4–8 2006 . New York: ACM Press, pp. 353–362.

Clark, Herbert H.; and Susan E. Brennan (1991). Grounding in communication. In L. Resnick, J. Levine, and S. Teasley (eds): *Perspectives on socially shared cognition*. Washington, DC: American Psychological Association, pp. 222–233.

Czerwinski, Mary; Edward Cutrell; and Eric Horvitz (2000). Instant Messaging and Interruption: Influence of Task Type on Performance. In C. Paris, N. Ozkan, S. Howard and S. Lu (eds): *OZCHI 2000. Proceedings of Interfacing Reality in the New Millennium*, Sydney, Australia, December 4–8 2000 . Sydney: CHISIG, pp. 356–361.

Dabbish, Laura; Colleen Stuart; Jason Tsay; and Jim Herbsleb (2012). Social coding in GitHub: transparency and collaboration in an open software repository. *CSCW'12. Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, Seattle, Washington, USA, February 11–15 2012 . New York: ACM Press, pp. 1277–1286.

Davies, Steven; and Marc Roper (2014). What's in a bug report? *ESEM'14. Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, Torino, Italy, September 18–19 2014 . New York: ACM Press, pp. 1–10.

de Souza, Cleidson R. B.; and David F. Redmiles (2011). The Awareness Network, To Whom Should I Display My Actions? And, Whose Actions Should I Monitor? *IEEE Transactions on Software Engineering*, vol. 37, no. 3, May 2011, pp. 325–340.

Erickson, Thomas; and Wendy A. Kellogg (2000). Social translucence: an approach to designing systems that support social processes. *ACM Transactions on Computer-Human Interaction*, vol. 7, no. 1, 2000, pp. 59–83.

Espinosa, J.; Sandra Slaughter; Robert Kraut; and James Herbsleb (2007). Team Knowledge and Coordination in Geographically Distributed Software Development. *Journal of Management Information Systems*, vol. 24, no. 1, 2007, pp. 135–169.

Greif, Irene; and D.R. Millen (2003). Communication Trends and the On-Demand Organization. *IBM - T.J. Watson Research Center - Cambridge, MA*, 2003.

Grudin, Jonathan (1994). Groupware and social dynamics: eight challenges for developers. *Communications of the ACM*, vol. 37, no. 1, January 1994, pp. 92–105.

Guo, Philip J.; Thomas Zimmermann; Nachiappan Nagappan; and Brendan Murphy (2011). Not my bug! and other reasons for software bug report reassignments. *CSCW'11. Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*, Hangzhou, China, March 19–23 2011 . New York: ACM Press, pp. 395–404.

Gutwin, Carl; Reagan Penner; and Kevin Schneider (2004). Group awareness in distributed software development. *CSCW'04. Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, Chicago, Illinois, USA, November 06–10 2004 . New York: ACM Press, pp. 72–81.

Halverson, Christine A; Jason B Ellis; Catalina Danis; and Wendy A Kellogg (2006). Designing task visualizations to support the coordination of work in software development. *CSCW'06. Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, Banff, Alberta, Canada, November 4–8 2006 . New York: ACM Press, pp. 39–48.

Handel, Mark; and James D. Herbsleb (2002). What is chat doing in the workplace? *CSCW'02. Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work*, New Orleans, Louisiana, USA, November 16–20 2002 . New York: ACM Press, pp. 1–10.

Herbsleb, James D.; David L. Atkins; David G. Boyer; Mark Handel; and Thomas A. Finholt (2002). Introducing Instant Messaging and Chat in the Workplace. *CHI'02. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Minneapolis, Minnesota, USA, April 20–25 2002 . New York: ACM Press, pp. 171–178.

Hupfer, Susanne; Li-Te Cheng; Steven Ross; and John Patterson (2004). Introducing collaboration into an application development environment. *CSCW'04. Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, Chicago, Illinois, USA, November 06–10 2004 . New York: ACM Press, pp. 21–24.

Jan, Syed Roohullan; Faheem Dad; Nouman Amin; Abdul Hameed; and Syed Saad Ali Shah (2016). Issues in global software development (communication, coordination, and trust): A critical review. *Journal of Software Engineering and Applications*, vol. 2, no. 2, 2016, pp. 660–663.

Jensen, Rasmus Eskild (2014). Why closely coupled work matters in global software development. *GROUP'14. Proceedings of the 18th International Conference on Supporting Group Work*, Sanibel Island, Florida, USA, November 09–12 2014 . New York: ACM Press, pp. 24–34.

Jensen, Rasmus Eskild; H. T. Storm; and Pernille Bjørn (2011). Global Software Development: The complexities in communicating about the requirement specification across culture and geography. In *Research Seminar on Information Systems (IRIS)*. Turku, Finland.

Jensen, Rasmus Eskild; and Pernille Bjørn (2012). Divergence and Convergence in Global Software Development: Cultural Complexities as Social Worlds. In *From Research to Practice in the Design of Cooperative Systems: Results and Open Challenges*. London: Springer.

Menendez-Blanco, Maria; Pernille Bjørn; Naja M. Holten Møller; Jesper Bruun; Hans Dybkjær; and Kasper Lorentzen (2018). GRACE: Broadening Narratives of Computing through History, Craft and Technology. *GROUP'18. Proceedings of the 2018 ACM Conference on Supporting Groupwork*, Sanibel Island, Florida, USA, January 07–10 2018 . New York: ACM Press, pp. 397–400.

Mi, Qing; and Jacky Keung (2016). An empirical analysis of reopened bugs based on open source projects. *EASE'16. Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, Limerick, Ireland, June 01–03 2016 . New York: ACM Press, pp. 1–10.

Miller, Matthew K; John C. Tang; Gina Venolia; Gerard Wilkinson; and Kori Inkpen (2017). Conversational Chat Circles. *CHI'17. Proceedings of the 2017 ACM SIGCHI Conference on Human Factors in Computing Systems*, Denver, Colorado, USA, May 06–11 2017 . New York: ACM Press, pp. 2394–2404.

Olson, Gary M; and Judith S Olson (2000). Distance matters. *Human-Computer Interaction*, vol. 15, no. 2, 2000, pp. 139–178.

Pinto, Danieli; Karoline Garcia; and Nelson Tenório (2017). Technological Communication Tools in Use - The Shape of Knowledge Shared within Software Development Teams. In K.Liu, J. Bernardino, A.C. Salgado and J. Filipe (eds): *IC3K 2017. Proceedings of the 9th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, Funchal, Madeira, Portugal, November 1–3 2017 . Setúbal, Portugal: SCITEPRESS, pp. 158–166.

Randall, Dave; Richard Harper; and Mark Rouncefield (2007). *Fieldwork for Design*: theory and practice. London: Springer-Verlag London Limited.

Saha, Ripon K.; Sarfraz Khurshid; and Dewayne E. Perry (2015). Understanding the triaging and fixing processes of long lived bugs. *Information and Software Technology*, vol. 65, 2015, pp. 114–128.

Schmidt, Kjeld; and Liam Bannon (1992). Taking CSCW seriously. *Computer Supported Cooperative Work (CSCW)*, vol. 1, no. 1–2, March 1992, pp. 7–40.

Schmidt, Kjeld; and Carla Simone (1996). Coordination Mechanisms: Towards a Conceptual Foundation of CSCW Systems Design. *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, vol. 5, no. 155, 1996, pp. 155–200.

Šmite, Darja; Nils Brede Moe; Aivars Šāblis; and Claes Wohlin (2017). Software teams and their knowledge networks in large-scale software development. *Information and Software Technology*, vol. 86, 2017, pp. 71–86.

Smith, S; and C Boldyreff (1995). Towards an Enterprise Modelling Method for CSCW Systems. *ISADS 95. Proceedings of the Second International Symposium Autonomous Decentralized Systems*, Phoenix, Arizona, USA, April 25–27 1995 . Los Alamitos, California: IEEE Computer Society, pp. 352–358.

Storey, Margaret Anne; Alexey Zagalsky; Fernando Figueira Filho; Leif Singer; and Daniel M. German (2017). How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development. *IEEE Transactions on Software Engineering*, vol. 43, no. 2, 2017, pp. 185–204.

Tan, Lin; Chen Liu; Zhenmin Li; Xuanhui Wang; Yuanyuan Zhou; and Chengxiang Zhai (2014). Bug characteristics in open source software. *Empirical Software Engineering*, vol. 19, no. 6, 2014, pp. 1665–1705.

Tsay, Jason; Laura Dabbish; and James Herbsleb (2014). Influence of social and technical factors for evaluating contribution in GitHub. *ICSE'14. Proceedings of the 36th International Conference on Software Engineering*, Hyderabad, India, May 31–June 07 2014 . New York: ACM Press, pp. 356–366.

Wang, Yi; and David Redmiles (2016). Cheap talk, cooperation, and trust in global software engineering. *Empirical Software Engineering*, vol. 21, no. 6, 2016, pp. 2233–2267.

Zahedi, Mansooreh; Mojtaba Shahin; and Muhammad Ali Babar (2016). A systematic review of knowledge sharing challenges and practices in global software development. *International Journal of Information Management*, vol. 36, no. 6, 2016, pp. 995–1019.

Zhao, Yangyang; Hareton Leung; Yibiao Yang; Yuming Zhou; and Baowen Xu (2017). Towards an understanding of change types in bug fixing code. *Information and Software Technology*, vol. 86, 2017, pp. 37–53.