Awareness and Merge Conflicts in Distributed Software Development

H.-Christian Estler* Martin Nordio* Carlo A. Furia* Bertrand Meyer*,†
*Chair of Software Engineering, Department of Computer Science, ETH Zurich, Zurich, Switzerland
†ITMO National Research University, St. Petersburg, Russia
firstname.lastname@inf.ethz.ch

***Abstract*—Collaborative software development requires programmers to coordinate their work and merge individual contributions into a consistent shared code base. Traditionally, coordination follows a series of "update-modify-commit" cycles, where merge *conflicts* arise upon committing if individual modifications have diverged and must be explicitly reconciled. Researchers have been suggesting that providing timely *awareness* information about "who's changing what" may not only help deal with conflicts but, more generally, improve the effectiveness of collaboration.**

This paper investigates the impact of awareness information in the context of globally distributed software development. Based on an analysis of data from 105 student developers constituting 12 development teams located in different countries, we analyze, among other things: 1) the frequency of merge conflicts and insufficient awareness; 2) the impact of distribution on team awareness; 3) the perceived impact of conflicts and lack of awareness on productivity, motivation, and project punctuality. Our findings include: 1) lack of awareness occurs more frequently than merge conflicts; 2) information about remote team members is missing roughly as often as information about co-located ones; 3) insufficient awareness information affects more negatively programmer's performance than merge conflicts.

***Index Terms*—Distributed software development, Empirical study, Awareness, Merge Conflicts**

I. Introduction

The inexorable trend of software development, from local to globally distributed, brings new kinds of challenges and aggravates existing ones—especially those related to *collaboration*, as globally distributed development is necessarily collaborative.

An elusive, yet major, issue in collaborative endeavors is *awareness*. Coordination between parties requires that they be aware of each other's work, to avoid duplications and redundancies, and to ensure that independently developed modules can function in combination. Conversely, insufficient awareness indicates lack of crucial information, which may disrupt progress and jeopardize efficiency and timeliness.

This paper presents an empirical study of the problems related to awareness deficiency in the practice of distributed software development, with the goals of understanding their severity and suggesting solutions to allay them or to prevent them from happening. A concrete simplified scenario will help better introduce the terms of the problem.

Motivating scenario. Anita and Bruno are developers who respectively belong to the American and Brazilian development units of a globally distributed software company; they have been working together on a software for trading stocks. As it is customary these days, they use a distributed version control system to keep track of changes and to share them. Following another good practice of software engineering, they have modularized the system so that Anita can work on improving the features to sell stocks while Bruno is working on a GUI for the whole system. Since they work on distinct private copies of the code base, Anita and Bruno can see each other's work only when they push the local commits to a shared repository. Bruno proceeds with his work until he realizes that he has been refactoring some functionality whose original design was Anita's. At this point, he has to interrupt his work and try to contact her. In an online meeting, they discuss progress in general terms; Anita is however reluctant to share her work so far, since she hasn't had a chance to test some new features that Bruno's work is using. After a few days, when they are finally ready to merge their changes to the code, it is clear that Bruno's refactoring work is inconsistent with Anita's. The two meet again online and reconcile the inconsistencies until merging is possible without conflicts. It turns out that Anita's conflicting changes were introduced over two weeks ago, but Bruno could not plan his work around them since he was unaware of the location and nature of Anita's changes.

Research questions. The scenario describes some frequently occurring problems related to insufficient awareness of the work of other developers. In particular, it highlights *merge conflicts* as a possible outcome, and hints at the fact that insufficient awareness often leads to work interruptions and deteriorates coordination. Based on this, we frame the problem of awareness into the following research questions:

- **RQ.A:** How frequent and how significant are merge conflicts? Do they originate more often in the work of co-located or of remote team members?
- **RQ.B:** How frequent and how significant is insufficient awareness (of the work of team members)? Does it originate more often in the work of co-located or of remote team members?
- **RQ.C:** What are the effects of merge conflicts and of insufficient awareness on project development?
- **RQ.D:** What are the frequency and detail level with which awareness information should be provided that are preferred by developers in a distributed setting?

Summary of findings. Section IV describes our study's findings in details. The most important results are:

- The likelihood of incurring into merge conflicts is not significantly affected by the location (co-located vs. remote) of developers within the same team.
- Interruptions due to insufficient awareness occur frequently for teams of non-trivial size. As for merge conflicts, the location of developers does not significantly affect the likelihood of such interruptions.
- Interruptions impact more negatively than just merge conflicts measures such as productivity, motivation, and keeping to the schedule.
- Developers would appreciate having access to awareness information frequently but not in real time; they have, however, diverse preferences regarding the level of detail in which such information should be made available.

Awareness applied to development tools. In the last decade, development tools have been introduced specifically to provide improved awareness of other developers' work, to reduce the chance that merge conflicts occur, and to facilitate resolution when conflicts do occur. The findings of this paper can help improve such tools by suggesting how awareness-related problems occur in practice and what their impact is. Therefore, Section II briefly reviews the state of the art for awareness tools, highlights their assumptions and further motivates the paper's empirical analysis.

II. Tools for Awareness and Merge Conflict Detection

We summarize related work, including ours, in developing awareness and conflict detection tools, to provide additional motivation for studying awareness and merge conflict problems independent of specific solutions and tools. Table I, which we reproduce from [10], gives a synoptic overview. Section VI reviews related work in other areas.

Awareness tools seek to raise developers' awareness about the changes introduced by others, with the goal of reducing conflicts and improving coordination. Awareness mostly targets direct (syntactic) conflicts and only simple indirect conflicts such as changes to method signatures. The granularity of details about what has changed varies from tool to tool, including line-based, class-based, and branch-based awareness. Question *RQ.D* of our empirical study investigates this aspect.

Advanced conflict detection. Conflict detection and conflict prediction tools search for conflicts and report them as soon as possible. These tools normally work by continuously trying to merge local copies of different users; whenever a speculative merge fails, it means that a conflict might occur. As shown in Table I, these tools may perform sophisticated merge conflict detection that include compilation and running tests. On the other hand, they usually do not provide information about how the code has been modified. Question *RQ.C* of our empirical study investigates how this information may affect developers.

Other tools for collaboration. A diversity of tools are used to simplify collaboration among distributed teams, including some commercial products such as IBM's Jazz [5] and Microsoft's Team Foundation [12]. Tools such as CodeRun [7] have brought IDEs to the web by largely replicating functionalities of traditional IDEs: every developer works on a different copy of the code, stored on a server. Some web-based IDEs, for example Cloud9 [6] and Collabode [13], support collaborative development through real-time code sharing: developers can simultaneously work on the same piece of code with the same view, as if they were editing a GoogleDoc shared document. Section IV-D suggests that this kind of collaboration is only useful in certain circumstances where direct tight interaction is required, such as in pair programming practices.

III. Design of the Empirical Study

The subjects participating in our empirical study were students enrolled in the 2013 edition of DOSE, a master-level university course on "Distributed and Outsourced Software Engineering" which we have been organizing for several years [22], [21]. Ten universities across the globe took part in the 2013 edition of the course: the University of Rio Cuarto (Argentina); the University of Adelaide (Australia); the Pontificia Universidade Catolica do Rio Grande do Sul (Brazil); the IT University of Copenhagen (Denmark); Cairo University (Egypt); the Politecnico di Milano (Italy); the State University of Nizhny Novgorod (Russia); the Universidad Politecnica de Madrid (Spain); ETH Zurich (Switzerland); and the University of Zurich (Switzerland).

DOSE teaches globally distributed software engineering in a globally distributed setting: its key component is a software development project carried out by the students in inter-university distributed teams. The 2013 project theme was the development of a platform for networked multi-player games. We defined the overall architecture so that it could accommodate top-level modules for different games. The 171 students from the 10 universities were arranged in 12 development teams; each development team produced a different game for the platform. 134 students were active developers; the remaining 37 students played the role of project customers.

Each team consisted of 3 groups with different responsibilities: one group focused on GUI and networking development; one group on implementing the game logic; and one group on the artificial intelligence component. The requirements elicitation phase was completed before the actual development started.[1] Each group consisted of 3 or 4 student developers; members of the same group were located in the same country and attended the same university. But the 3 different groups making up a development team were located in different countries, so as to make it a distributed development effort.

For our empirical study, we collected data about the development that spread over a total of 4 weeks, and produced 12 successful projects. (The average project size was 11'776 lines of code in 68 classes).

[1]For organizational reasons that are not important here, it mainly involved students from Australia and Brazil.

	direct conflict	indirect conflict	false positives	availability	Awareness			Editing	
					line-based	class-based	branch-based	shared editing	automatic merging
FastDash [2]	no	no	–	real-time	yes	yes	yes	no	no
Palantir [26]	detect	detect	–	real-time	no	yes	no	no	no
Syde [19]	detect, inspect	no	no	real-time	no	yes	no	no	no
CollabVS [16]	detect	detect, inspect	yes	real-time	yes	yes	no	no	no
Crystal [4]	detect	detect	no	commit	no	no	yes	no	no
WeCode [15]	detect	detect	no	saving	no	no	yes	no	no
CloudStudio [10]	prevent, detect, inspect	detect, inspect	no	real-time	yes	yes	yes	no	yes
Collabode [14]	no	no	–	no	no	no	no	yes	no
Cloud9 [6]	no	no	–	no	no	no	no	yes	no

TABLE I: MAIN FEATURES OF AWARENESS SYSTEMS AND CONFLICT DETECTION TOOLS. For each system, we report: whether it supports detection of *direct conflicts* and of *indirect conflicts* (cf. [26]); whether conflict reports may include *false positives*; whether conflicts are *available* in real-time or upon commit; the granularity of the *awareness* system (line, class, or branch); whether collaborative *editing* supports shared sessions a la Google Doc and automatic merging of versions.

A. Data Collection

We collected data for our study from two sources:[2] a questionnaire and a real-time tracking tool.

While merge conflicts are well-defined events, a general problem with studies of awareness is that it is hard to measure lack of or insufficient awareness directly. Instead, we shifted all questions from awareness to the tightly related—but much more clearly defined and tangible—problem of *interruptions* due to lack of or inconsistent information. For example, if we cannot proceed because we realize that a function (originally developed by another team member) does not behave as per requirements, it would be useful to acquire the information about whether other developers have noticed the same problem, and whether someone is already in the process of fixing it. Thus, we make the underlying assumption that: *interruptions of the development workflow are indicative of insufficient awareness of the work of other developers on the same team*. Section IV-C will discuss data that confirms a posteriori the soundness of this assumption.

Questionnaire. The questionnaire consisted of two parts. The first part asked for a self-assessment of the participants' experience in distributed development and in some of the tools used for the project. Section III-B outlines the picture that emerged from these questions. The second, and major, part of the questionnaire asked the participants to express their preferences and their experience during project development regarding several aspects:

- Merge conflicts and usage of distributed version control (specifically, Git).
- Interaction and problems during development with the local (intra-group) team members.
- Interaction and problems during development with the remote (inter-group) team members.
- How they coped with interruptions due to interaction.
- Preferences regarding awareness of other team members' work.

We illustrate the detailed content of the specific questions as we analyze the data in Section IV.

Real-time tracking. With the goal of collecting finer-grained quantitative information about interruptions of the workflow (as proxies for awareness problems), we built a web-based session tracking tool called *James*.[3] Whenever starting a programming session, one can also activate James in a browser. James provides a simple interface to enter data about interruptions, shown in Figure 1. Whenever a workflow interruption occurs, one increments the counter according to the nature of the problem and to whether the object of the interruption pertains the work of a local or remote team member. James logs all the events in real time, and also offers the option to store free text notes in case something unusual is worth recording.

B. Participants

Questionnaire. At the end of the course, we invited all 134 student developers of DOSE 2013 to fill in the questionnaire; no reward was offered. 105 of the students accepted the invitation to fill in the questionnaire, giving a solid response rate of 78%.

Background and experience. It is useful to understand the self-assessed experience of the 105 participants in matters of distributed development. We asked to rate their experience on a scale of 1 to 7 (where 1 means *no experience* and 7 means *lots of experience*) in: programming in general; programming in

[2]The questionnaire and all collected data is available for analysis and replication: http://se.inf.ethz.ch/data/awareness.

[3]Available at http://cloudstudio.ethz.ch/james.

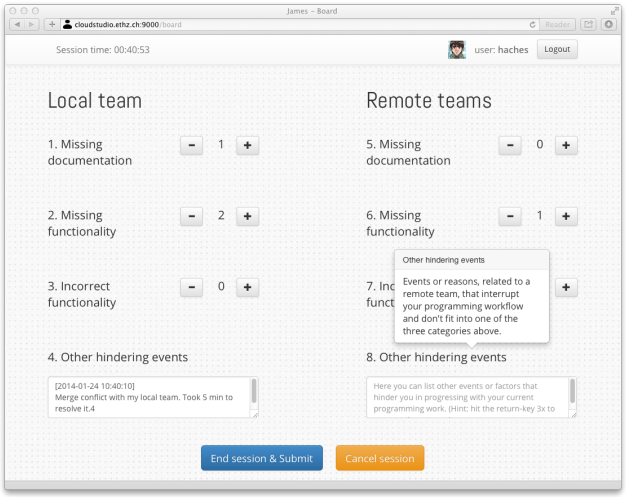

Fig. 1: Screenshot of the web-based tracking tool.

teams of two or more developers; programming in *distributed* teams of two or more developers; and using distributed version control (in particular, Git). Figure 2 and Table II summarize the results. Overall, the participants had fairly extensive (for students) experience in programming and team programming, but little or no experience with distributed development and distributed version control systems. As we will see in Section IV, some of the study results reflect this situation (in particular, the lack of experience with Git exacerbated some problems related to merge conflicts).

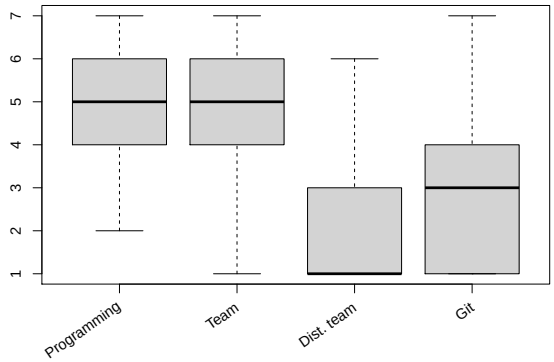

Fig. 2: Experience of the study participants in programming, team programming, distributed team programming, and Git. Experience ranges over a 1–7 scale.

Real-time tracking. Before beginning the development phase, we invited all the 134 student developers of DOSE 2013 to use the real-time tracking tool during development; no reward was offered other than a public acknowledgment. Since this was a much more burdensome task than filling in a questionnaire, we expected only a minority of motivated students to reply; we made clear in the request that there was no obligation involved, but also that we expected volunteers

experience	median	min	max	mean
programming	5	2	7	4.6
team programming	5	1	7	4.5
distributed programming	1	1	6	2.1
Git	3	1	7	2.9

TABLE II: Experience of the study participants in programming, team programming, distributed team programming, and Git. Experience ranges over a 1–7 scale.

to do a good job and use the tool throughout the 4 weeks of development. 15 students accepted the invitation and used the real-time tracking tool. Post mortem, we removed the few obviously spurious sessions from the logs, leaving us with data about 106 distinct development sessions, each lasting an average of 2.9 hours. Table III shows more statistics about the data collected with the session tracking tool.

# developers	15
# programming sessions	106
# sessions with interrupt	36
total development time (hrs)	311
median session time (hrs)	2
mean session time (hrs)	2.9
median sessions per user	5
mean sessions per user	7
min sessions per user	1
max sessions per user	34

TABLE III: Data collected through the tracking tool.

We discuss in detail the data about real-time tracking in combination with the questionnaire data in the next section.

IV. Results of the Empirical Study

The data collected in our empirical study demonstrates how merge conflicts and (lack of) awareness impact on distributed software development activities—in particular, coordination between developers on the same team. This section presents the results that emerged from a combined analysis of the questionnaires and the real-time tracking data (described in Section III). Sections IV-A through IV-D describe the main findings corresponding to research questions *RQ.A* through *RQ.D*; Section IV-E discusses correlations between data and how they delineate other potentially disruptive aspects of distributed software development.

A. Merge Conflicts: Frequency and Origin

Research question *RQ.A* points to the well-known problem of merge conflicts: upon trying to export their local modifications into a shared global code base, developers may introduce inconsistent changes that must be reconciled. This section discusses how frequently merge conflicts occurred in our study; how much time conflict resolution took; and whether changes by co-located or by remote team members were the main source of conflicts.

Frequency. To get a qualitative picture of how frequently merge conflicts occur, we asked how many conflicts occurred

over the four weeks of project development; Table IV summarizes the questionnaire responses. Overall, more than 94% of the 105 developers had to deal with some merge conflicts, but only a minority experienced them very frequently.

number of conflicts	responses	%
0	7	6.7
1–4	61	58.1
5–9	19	18.1
≥ 10	18	17.1

TABLE IV: Number of merge conflicts encountered during a four-week development period.

Significance. To get an idea of the severity of merge conflicts, we asked how much time developers spent to resolve a single conflict, on average and in the worst case; Table V summarizes the questionnaire responses. Conflicts do not normally seem to take a lot of time to resolve: nearly 70% of the developers spent no more than 10 minutes to deal with a conflict on average. However, the worst cases of conflict resolution can be substantially more time consuming.

	average case		worst case	
time (min)	**responses**	**%**	**responses**	**%**
≤ 1	4	4.3	1	1.1
1–5	28	30.1	11	11.8
5–10	33	35.5	17	18.3
10–20	13	14	19	20.4
≥ 20	15	16.1	45	48.4

TABLE V: Average- and worst-case time (in minutes) spent to resolve one merge conflict.

Origin. To estimate whether merge conflicts are more frequently due to the parallel work of co-located or of remote team members, we asked whether "most conflicts" occurred: with *local* (i.e., co-located) members of your team; with *remote* members of your team; or with developers of other teams. The last option was meant to indicate cases of unwanted interaction between the work of independent teams, possible since all teams committed into different directories and branches but in the same physical repository. Table VI summarizes the questionnaire responses. The difference between local and remote team members is not large and probably not significant[4]; this may indicate that the likelihood of conflicts depends more directly on other factors than location. More significantly, the majority of responses indicate the work of other teams as the main source of conflicts suggesting that bad practices in the usage of the Git version control system were the norm rather than the exception. Our intent in using a unique repository was to let developers have read access to the work of every other teams, while using the powerful features of Git to manage their work independently. Unfortunately, the data indicates that this didn't work in practice as expected; the result is consistent with the fact that most developers indicated a below average experience with Git and distributed version control (see Section III-B). Overall, our study did not find major differences between local and remote team members as origins of source conflicts; but the project setup decreases our confidence in this appraisal.

[4]Statistical significance tests do not seem to be applicable: paired tests (such as the Wilcoxon signed-rank test) typically require samples of the same population in different conditions, and unpaired two-sample tests (such as the U test) typically require independent samples. Neither seems justifiably the case in our experiments of local vs. remote: local and remote conditions in each pair are reported by the same person (hence they are likely dependent); the populations of local and of remote are not directly comparable (in our setup, each local group interacts with two remote groups).

most conflicts with:	responses	%
local team members	30	30.9
remote team members	22	22.7
members of other teams	45	46.4

TABLE VI: Origin of most conflicts: local vs. remote team members.

Merge conflicts will occur but are not very frequent; most of them have limited significance, as they take few minutes to resolve. The location of team member does not seem to have a major effect on the likelihood that their work will introduce conflicts.

B. Workflow Interruptions: Frequency and Origin

Research question *RQ.B* looks for problems related to insufficient awareness of the work of team members. It is hardly possible to measure (lack of) awareness in real-time, since it is a somewhat elusive notion that emerges mostly as an afterthought: for example, in the scenario outlined in Section I, a merge conflict makes Bruno realize that he *previously* was unaware of Anita's work.

Given this nature of awareness, we collected indirect data about possible *effects* of the lack of awareness in terms of *interruptions* caused to a developer's workflow: due to lack of documentation about some functionality; due to some expected functionality missing altogether; and due to functionality present but incorrect (that is, behaving differently than documented). The questionnaire asked to estimate the frequencies of these interruptions; and the real-time tracking system recorded these interruptions as they were flagged.

Frequency. To get a qualitative picture of how frequently workflow interruptions (as a sign of awareness deficiencies) occur, we asked to assess their frequencies on an ordinal scale from 1 to 7 (where 1 means *never* and 7 means *very often*). Table VII summarizes the questionnaire responses, which were broken down according to kind of interruption (missing documentation, missing functionality, and incorrect functionality) as well as to whether the source of interruptions was related to the work of local or of remote team members. Overall, interruptions to one developer's workflow seem to be roughly proportional to the number of team members he or she interacts with, as each local group collaborated with two remote groups (and all groups were roughly the same

size). This entails that the frequency of interruptions becomes significant (but hardly overwhelming) as soon as more than few people collaborate closely on the same project.

interruption		median	min	max	mean
documentation	L	2	1	7	2.5
	R	4	1	7	3.7
functionality	L	2	1	7	2.4
	R	5	1	7	4.4
incorrect	L	2	1	7	2.3
	R	4	1	7	3.9

TABLE VII: Frequency of workflow interruptions according to the questionnaire responses. There are three kinds of interruptions: due to missing *documentation*, missing *functionality*, and *incorrect* functionality. The source of interruptions can be the work of local (L) or remote (R) team members. Frequencies range over a 1–7 scale.

To quantitatively look at interruption frequency, we asked to tag interruptions in each category in real time. Table VIII summarizes the data about the 15 developers who used the real-time tracking tool over a total of 106 sessions, each lasting 2.9 hours on average. Overall, interruptions occurred in 34% of the 106 sessions; assuming that our sample is representative of the actual distribution of interruptions, we expect a workflow interruption every 2.5 hours. These figures are consistent with the qualitative findings, and suggest that interruptions occur with significant frequency for teams of non-trivial size.

interruption		#	ratio R/L
documentation	L	9	1.9
	R	17	
functionality	L	25	1.6
	R	39	
incorrect	L	16	1.2
	R	19	
TOTAL	L	50	1.5
	R	75	

TABLE VIII: Number of workflow interruptions recorded in real time by 15 developers working for a total of 307 hours. There are three kinds of interruptions: due to missing *documentation*, missing *functionality*, and *incorrect* functionality. The source of interruptions can be the work of local (L) or remote (R) team members.

Kind. The collected data suggests that interruptions due to *missing functionality*—that is, a functionality described in the requirements but not available in the code base—are the most frequent kind among the three considered (missing documentation, missing functionality, and incorrect functionality). Overall, problems with functionality (missing or incorrect) account for nearly 80% of the interruptions according to the real-time tracking data. This is consistent with the emphasis on the requirements elicitation phase in project development, where we insisted that teams agree on a reasonable requirements specification document before they start the implementation phase.

Origin. As we mentioned when discussing interruption frequencies, remote team members are twice as numerous as local team members. Therefore, *if location plays no dominant role in determining interruptions*, it is to be expected that the *origin* of interruptions be the work of remote team members more frequently than the work of local team members. This is what we observe in Table VII, where median and means for remote team members are consistently higher than the corresponding measures for local team members. To have a visual confirmation of this difference, Figure 3 shows the distribution of answers summarized in Table VII: the distributions for co-located team members are slanted towards low frequencies, whereas those for remote team members are more uniform and include higher frequencies. The quantitative data of Table VIII confirms this trend: remote interruptions are overall 50% more numerous than local ones.

If we refine the picture by weighing in additional factors, we can understand the fine-grain differences between local and remote shown in Table VIII. First, while remote team members are roughly twice as numerous as local ones, the modular architecture of the systems and the matching division of labor were such that local team members interacted more closely on the same piece of code. Second, problems related to functionality (missing or incorrect) determine interruptions that are likely more prominent than missing documentation: in the latter case, one can often resort to exploration and testing to understand the behavior; in contrast, if functionality belonging to another component is missing or incomplete, one probably has to check with the responsible developer what the issue really is. Therefore, we expect the closer interaction associated with local team members to affect more strongly interruptions related to functionality. These observations help explain the data in Table VIII: location is essentially irrelevant for missing documentation (where an almost perfect 2-to-1 ration of interruptions is observed); whereas it affects problems related to functionality to some extent, as a result of local team members interacting more closely on the same module.

Workflow interruptions (related to insufficient awareness) occur fairly frequently for teams of non-trivial size. Interruptions due to functionality (missing or incorrect) are more frequent than interruptions due to documentation problems. Developer location seems to play no major role in determining the frequency of interruptions.

C. Conflicts and Interruptions: Impact and Effects

Research question *RQ.C* asks for the *effects* of merge conflicts and of insufficient awareness on project development. To address these issues, we asked to estimate the *impact* of merge conflicts and of workflow interruptions (which we use as a signal of insufficient awareness) on various dimensions of success; and to describe what *actions* are normally undertaken when workflow is interrupted.

Impact. We asked to assess the negative impact of both merge conflicts and workflow interruptions on one's pro-

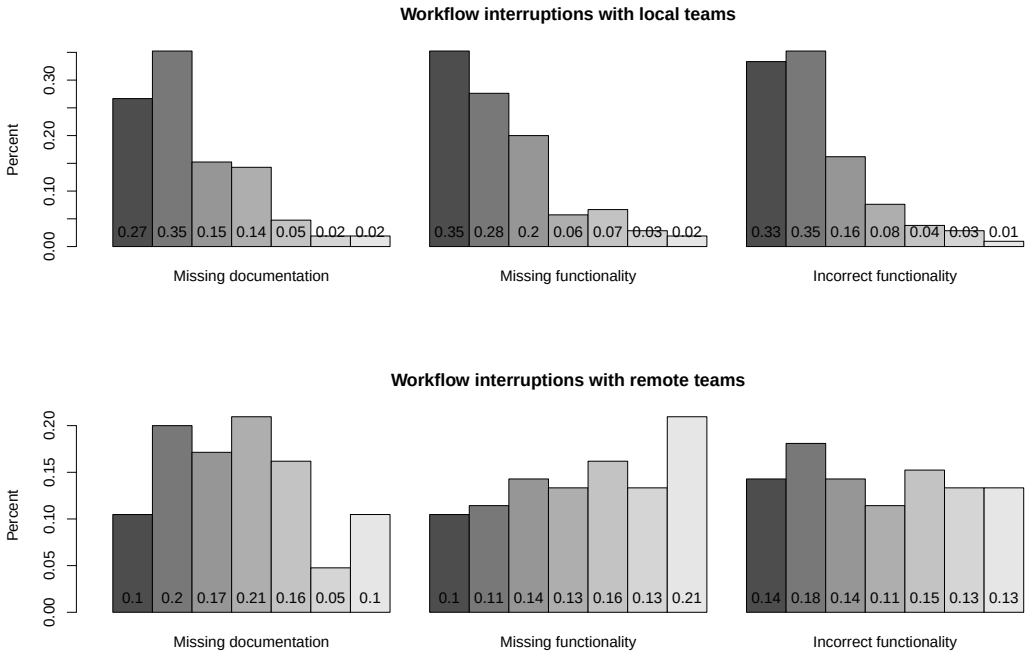

Fig. 3: Frequency of workflow interruptions according to the questionnaire responses. The top row shows interruptions caused by the work of local team members; the bottom row shows interruptions caused by the work of remote team members. Frequencies range over a 1–7 scale.

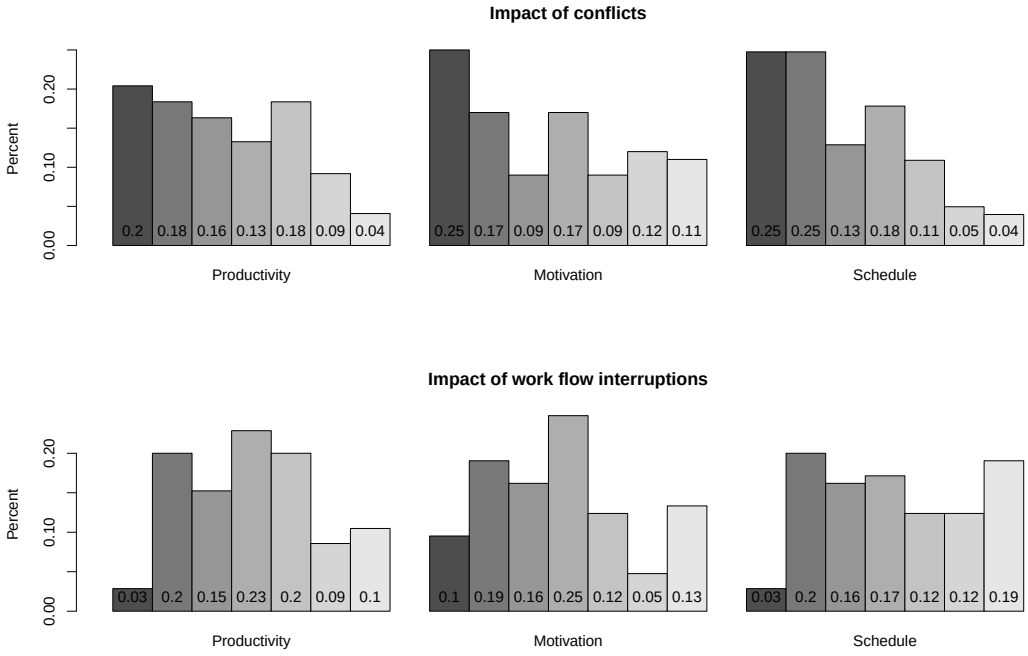

Fig. 4: Impact of conflicts (top row) and of workflow interruptions (bottom row) on *productivity*, *motivation*, and ability to keep to the assigned *schedule*. Impact ranges over a 1–7 scale, where higher rank means more negative impact.

ductivity, motivation, and keeping to the schedule using a scale of 1 to 7 (where 1 means *had no negative impact at all* and 7 means *had very much negative impact*). Table IX summarizes the questionnaire's responses; Figure 4 details the distribution of responses. While there seems to be no significant difference between the effects on productivity, motivation, and keeping to the schedule, there is a systematic bias in favor of merge conflicts; that is, the negative impact of merge conflicts is consistently lower than the negative impact of workflow interruption. This is confirmed (with different confidence in different categories) by a Wilcoxon signed-rank test[5] comparing conflicts and interruptions in each category, whose p-values are given in the last column. This data suggests that interruptions that are ultimately related to awareness problems are more pernicious or however have a broader impact than "run-of-the-mill" merge conflicts.

impact of	on	median	min	max	mean	p
MC	productivity	3	1	7	3.3	0.001
WI		4	1	7	4.0	
MC	motivation	3	1	7	3.5	0.08
WI		4	1	7	3.8	
MC	schedule	3	1	7	3.0	10^{-7}
WI		4	1	7	4.3	

TABLE IX: Negative impact of merge conflicts (MC) and workflow interruptions (WI) on: productivity, motivation, and keeping to the schedule. Impact ranges over a 1–7 scale.

Effects. To get an idea of how workflow interruptions are dealt with, we asked to indicate one or more actions typically done when the workflow is interrupted (due to lack of awareness). The questionnaire included four predefined answers: "I check the repository for a recent commit"; "I contact the responsible developer asynchronously (e.g., via email)" ; "I contact the responsible developer in real time (e.g., via Skype)"; "I switch to a different task that does not depend on the interruption"; as well as the option to write down an open answer. The open answer option was not used by the respondents; Table X summarizes the results for the predefined answers. The least popular option is switching to a different task; this indicates that most workflow interruptions require to be dealt with by acquiring missing information and cannot be worked around by adjusting the workflow. This is an indirect confirmation of our assumption that workflow interruptions are often due to missing awareness; and that tools enhancing the awareness of other developers' work may help reduce disruptive interruptions.

Workflow interruptions (related to insufficient awareness) have a more negative impact than merge conflicts on productivity, motivation, and keeping to the schedule. Most interruptions require acquisition of missing information to continue, and thus disrupt the planned workflow.

D. Awareness: Preferences and Granularity

The answers to *RQ.C* in Section IV-C suggest that workflow interruptions are a serious issue and originate in insufficient or missing awareness information. Research question *RQ.D* investigates the follow-up problem of whether developers would welcome tools that make awareness information available more often than it is available using standard tools. To answer this question, we first investigate the overall preference for or against awareness information; then we try to figure how frequently and with what detail awareness information should be made available.

[5]In this case, data in each pair are from comparable populations.

action	responses	% of respondents
check repository	67	64.4
contact via email	52	50.0
contact via Skype	46	44.2
switch task	24	23.1

TABLE X: Actions developers taken when their workflow is interrupted due to missing/incorrect functionality or documentation. Developers could choose more than one option.

Overall preference. We asked to assess how helpful having access to information about the work of other developers before they push to the repository would be. Helpfulness ranges on an ordinal scale from 1 to 7 (where 1 means *not helpful at all* and 7 means *very helpful*). Table XI summarizes the answers. Overall, there is some preference but not a very strong one.

	median	min	max	mean
How helpful?	1	4	7	3.81

TABLE XI: How helpful would it be to have awareness information? Usefulness ranges over a 1–7 scale.

Time granularity. Given that lack of awareness is a significant source of nuisances and problems (Section IV-C), we partly attribute the limited enthusiasm for awareness information to the difficulty of evaluating a fairly abstract idea. Developers have a hard time imagining exactly how awareness information would be displayed and made available, which would determine whether it would be helpful.

To get into more concrete options, we asked to choose one or more preferred "times" when information about other developers in one's own team should be displayed: in real-time as if having complete access to the other developers' computers; whenever a developer completes a successful compilation; whenever a developer commits locally; or whenever a developer pushes committed changes to the shared repository. Table XII summarizes the answers. The option of being notified whenever a developer in one's team successfully completes a version of the system is the most popular one, chosen by over half of the respondents. In contrast, being notified in real time as developers type in their changes is the least popular option. Overall, the responses reinforce the timid preference for awareness information indicated by the previous generic question. They also suggest that real-time awareness can be distracting rather then helpful: the real-time collaboration features highlighted by web-based IDEs such as

those discussed in Section II are probably mostly useful only for practices that require a tight interaction between very few developers (such as remote pair programming), but would not scale to larger teams and more complex interactions.

when?	responses	% of respondents
real time	28	26.7
compile time	57	54.3
commit time	34	32.4
push time	39	37.1

TABLE XII: When should awareness information be made available? Developers could choose more than one option.

Detail granularity. Having ascertained that awareness information is helpful to have as long as it's not displayed too frequently, we assessed the orthogonal dimension of "detail". We asked to choose one or more preferred detail levels for the information about other developers in your team: the "full detail" of all changes and additions to the code base; which routines have been affected (but now how they have been modified); which classes have been affected (but not which routines within those classes); which packages or other modules have been affected (but not which classes within those packages); and no information about the change at all. Table XIII summarizes the answers. We observe varied preferences, but with a trend towards more detail. Overall, this suggests that as much information as possible should be available within an awareness tool, but it should also be possible for users to customize what is displayed according to their individual preferences of the moment: one size does not fit all.

what?	responses	% of respondents
full detail	37	35.2
changed routines	32	30.5
changed classes	25	23.8
changed packages	5	4.8
no information	6	5.7

TABLE XIII: What should the detail of awareness information be? Developers could choose more than one option.

Most developers prefer awareness information to be available at significant events (such as successful compilation) rather than in real time. Different developers have different preferences regarding the level of detail that awareness information should have.

E. Correlation Analysis

To look for dependencies between responses, we performed a correlation analysis between all pairs of variables corresponding to the answers to the questionnaire, including those for the self-assessment of the respondents' experience with programming and collaboration (see Section III-B). Table XIV shows all variable pairs that exhibit a significant correlation.

The correlations are not particularly surprising in hindsight. For example, the correlations between the number of interruptions due to incorrect and missing functionality (both with local and with remote team members) reflect the fact that well-coordinated teams (with well-understood requirements) will have fewer problems with both incorrect and missing functionality, whereas badly-coordinated teams encounter problems with both. On the other hand, *missing* correlations bring more insight. Specifically, the lack of significant correlations with variables measuring the level of experience with distributed version control and other techniques for collaborative software development suggests that the findings distilled from the questionnaire are not a direct result of the background and pre-existing skills of the respondents. This reflects positively on the generalizability of our findings.

V. Threats to Validity

Internal validity. To minimize threats to internal validity, we manually went through all data (105 questionnaire responses, and 106 real-time sessions from 15 subject) looking for obviously inconsistent or bogus information. For example, we discarded the data of respondents that declared that they "never experienced conflicts" and took "over 20 minutes to resolve conflicts on average"; and very short tracking sessions that increased and decreased the interruption counts without sensible pattern. The fact that only 8% of the data points had to be discarded suggests a fundamentally good quality of the sanitized data. We also reran our analysis on data that only contained the top-50 developers (in number of commits). All the qualitative findings applied to this subset as well. This gives us confidence that our findings are largely independent of the development time invested by participants.

External validity. The major threat to generalizability comes from the majority of participants' low level of experience with Git, which is something not to be expected of experienced professional developers. Nevertheless, in spite of their limited experience, our study participants did not find merge conflicts an overall dominant problem; therefore, it is reasonable to speculate that experienced developers would strengthen our findings about the severity of workflow interruptions. Another threat comes from only involving student developers, albeit at the master's level. The fact that the results showed no correlation linking developers' experience to the significant measured variables indicates, however, that this threat is unlikely to be severe.

VI. Related Work

Other challenges and issues in globally distributed development have been investigated empirically; for example, the effect of time zones [17], [9], [20]; the role of development processes [11]; the impact on productivity and quality [24], [3]; the usage of contracts [23]; and the role of dispersion [25].

There is a broad body of research about awareness in distributed software development [27], mostly targeting aspects related to project management—orthogonal to those of the present paper which focuses on the practical impact on development activities. [1], [18], for example, focus on

correlation of	and of	τ	p-value
average time to resolve conflict	worst time to resolve conflict	0.75	0
incorrect due to Local	functionality due to Local	0.57	10^{-12}
documentation due to Remote	incorrect due to Remote	0.60	10^{-15}
functionality due to Remote	incorrect due to Remote	0.55	10^{-13}
MC impact on motivation	MC impact on productivity	0.55	10^{-12}
MC impact on schedule	MC impact on productivity	0.56	10^{-12}
WI impact on motivation	WI impact on productivity	0.61	10^{-15}
WI impact on schedule	WI impact on productivity	0.66	0
MC impact on motivation	WI impact on motivation	0.53	10^{-11}

TABLE XIV: All variable pairs that correlate (Kendall's $\tau > 0.5$) with high significance ($p < 10^{-3}$).

high-level aspects such as personal information or level of expertise; similarly, [8] shows the importance for developers of information about requirements, planning, and project status.

Research on awareness tools, reviewed in Section II, is often accompanied by experiments to test the tools against traditional version control systems. These experiments normally involve [16], [10], [26] small (2 or 3 people) teams working on controlled artificial exercises (such as refactoring) that can be carried out in a short programming session. In other cases [4], the experiments used regression data from open-source software repositories to estimate the effectiveness of their conflict detection mechanisms. The present paper's study complements both kinds of experiments by targeting real software development over a considerable time span, beyond the information stored by version control systems, and independent of specific solutions (as they were implemented in the evaluated tools).

VII. Conclusions

This paper presented an empirical study of the significance of merge conflicts and awareness-related problems in distributed software development. To achieve general findings independent of specific tool protocols, we studied interruptions to the workflow as proxies for situations of insufficient awareness. The findings confirm the benefits of improving awareness information; suggest that different developers often have different preferences regarding the frequency and detail that awareness information should have; and do not indicate developer location as a major factor in determining merge conflicts or awareness problems.

References

[1] G. Aranda, A. Vizcaino, R. Palacio, and A. Moran. What information would you like to know about your co-worker? A case study. In *ICGSE*, pages 135–144, 2010.

[2] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson. FASTDash: A visual dashboard for fostering awareness in software teams. In *CHI*, pages 1313–1322. ACM, 2007.

[3] 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 *ICSE*, pages 518–528. IEEE, 2009.

[4] Y. Brun, R. Holmes, M. Ernst, and D. Notkin. Proactive detection of collaboration conflicts. In *ESEC/FSE*, pages 168–178. ACM, 2011.

[5] L.-T. Cheng, C. R. de Souza, S. Hupfer, J. Patterson, and S. Ross. Building collaboration into IDEs. *ACM Queue*, 1(9):40–50, 2003.

[6] Cloud9 IDE. http://www.cloud9ide.com.

[7] CodeRun Studio. http://www.coderun.com.

[8] K. Dullemond and B. van Gameren. What distributed software teams need to know and when: An empirical study. In *ICGSE*, pages 61–70, 2013.

[9] J. A. Espinosa, N. Nan, and E. Carmel. Do gradations of time zone separation make a difference in performance? A first laboratory study. In *ICGSE*, pages 12–22. IEEE, 2007.

[10] H.-C. Estler, M. Nordio, C. A. Furia, and B. Meyer. Unifying configuration management with awareness systems and merge conflict detection. In *ASWEC*, pages 201–210. IEEE, 2013.

[11] H.-C. Estler, M. Nordio, C. A. Furia, B. Meyer, and J. Schneider. Agile vs. structured distributed software development: A case study. In *ICGSE*, pages 11–20. IEEE, 2012.

[12] M. T. Foundation. http://www.microsoft.com/visualstudio/en-us/products/2010-editions/team-foundation-server/overview.

[13] M. Goldman, G. Little, and R. C. Miller. Collabode: Collaborative coding in the browser. In *CHASE*, pages 65–68. ACM, 2011.

[14] M. Goldman, G. Little, and R. C. Miller. Real-time collaborative coding in a web IDE. In *UIST*, pages 155–164. ACM, 2011.

[15] M. L. Guimarães and A. R. Silva. Improving early detection of software merge conflicts. In *ICSE*, pages 342–352. IEEE Press, 2012.

[16] R. Hegde and P. Dewan. Connecting programming environments to support ad-hoc collaboration. In *ASE*, pages 178–187. ACM, 2008.

[17] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter. Distance, dependencies, and delay in a global collaboration. In *CSCW*, pages 319–328. ACM, 2000.

[18] Z. U. R. Kiani, D. Smite, and A. Riaz. Measuring awareness in cross-team collaborations – distance matters. In *ICGSE*, pages 71–79, 2013.

[19] M. Lanza, L. Hattori, and A. Guzzi. Supporting collaboration awareness with real-time visualization of development activity. In *CSMR*, pages 202 –211, 2010.

[20] M. Nordio, H.-C. Estler, B. Meyer, J. Tschannen, C. Ghezzi, and E. Di Nitto. How do distribution and time zones affect software development? A case study on communication. In *ICGSE*, pages 176–184. IEEE, 2011.

[21] M. Nordio, C. Ghezzi, B. Meyer, E. Di Nitto, G. Tamburrelli, J. Tschannen, N. Aguirre, and V. Kulkarni. Teaching software engineering using globally distributed projects: the DOSE course. In *CTGDSD*, pages 36–40. ACM, 2011.

[22] M. Nordio, R. Mitin, and B. Meyer. Advanced hands-on training for distributed and outsourced software engineering. In *ICSE*, pages 555–558. ACM, 2010.

[23] M. Nordio, R. Mitin, B. Meyer, C. Ghezzi, E. Di Nitto, and G. Tamburelli. The role of contracts in distributed development. In *SEAFOOD*, volume 35 of *LNBIP*. Springer-Verlag, 2009.

[24] N. Ramasubbu and R. Balan. Globally distributed software development project performance: An empirical analysis. In *ESEC/FSE*, pages 125–134. ACM, 2007.

[25] N. Ramasubbu, M. Cataldo, R. K. Balan, and J. D. Herbsleb. Configuring global software teams: A multi-company analysis of project productivity, quality, and profits. In *ICSE*, pages 261–270. ACM, 2011.

[26] A. Sarma, G. Bortis, and A. van der Hoek. Towards supporting awareness of indirect conflicts across software configuration management workspaces. In *ASE*, pages 94–103. ACM, 2007.

[27] R. L. Vivian, E. H. M. Huzita, G. C. L. Leal, and A. P. C. Steinmacher. Context-awareness on software artifacts in distributed software development: A systematic review. In *CRIWG*, pages 30–44. Springer, 2011.