J Braz Comput Soc (2012) 18:201–211
DOI 10.1007/s13173-012-0067-5

ORIGINAL PAPER

Analyzing the scalability of coordination requirements of a distributed software project

Cleidson R.B. de Souza · Jean M.R. Costa · Marcelo Cataldo

Received: 27 January 2011 / Accepted: 21 February 2012 / Published online: 3 April 2012

Abstract Collaborative tools have been proposed to support different domains, including software development. Despite important previous work on the design of collaborative tools, none directly addresses the required scalability of these collaborative tools. As such, the design of these tools is hindered because it does not take into account real-world requirements for handling and presenting information to support collaborative activities. In this paper, we use an approach to compute the coordination requirements of an actor, and by so doing, we are able to identify the required scalability of the collaborative tools to support these actors. The coordination requirements are the likely set of actors that a particular individual might need to coordinate/communicate with. We computed the coordination requirements of software developers involved in a large-scale distributed software development project. Software developers' coordination requirements were computed in four different time intervals: 1, 7, 15, and 30 days so that we could assess the scalability of collaborative tools in both short and long term software development activities. Our results suggest that, in some cases, the number of coordination requirements of a given actor might be very large, and that current collaborative software development tools do not provide proper support for such values. Furthermore, we observed that coordination requirements often involved members of different teams and from different locations, which increases the importance of having collaborative tools properly designed, given the difficulty of collaborating across organizational and geographical boundaries.

Keywords Scalability · Coordination requirements · Empirical study · Distributed software development

1 Introduction

All activities involving groups of actors working together require that these actors coordinate their activities to guarantee that the contributions made by an actor do not affect the contributions from other actors. In software development, this is not different: to ensure a proper integration of software components, the work of all developers needs to be aligned [21]. In order to do that meetings, emails, informal conversations, and other approaches [1, 30, 39] are essential. In addition, tool support plays an important role in facilitating the coordination of software development activities [1, 16, 21, 24]. For instance, configuration management systems allow developers to be aware of their colleagues' activities and align their work accordingly [21]. In particular, collaborative software development tools help developers to avoid errors [16, 34, 37], find experts [32], identify social and technical dependencies [42], among other advantages.

Given the benefits of collaborative software development tools, it is no surprise that these tools have been built by different researchers [6, 14, 15, 32, 36, 37, 42]. A limitation of

This is a revised and extended version of a previous paper that appeared at SBSC 2010, the Brazilian Symposium on Collaborative Systems.

C.R.B. de Souza (✉) · J.M.R. Costa
Faculdade de Computação, Universidade Federal do Pará, Belém, Brazil
e-mail: cleidson.desouza@acm.org

J.M.R. Costa
e-mail: jeanmrc@gmail.com

M. Cataldo
Institute for Software Research, Carnegie Mellon University, Pittsburgh, USA
e-mail: mcataldo@cs.cmu.edu

these tools is that they are designed for small teams [38], therefore, it is not possible to ensure that they can handle large amounts of information. For instance, tools that use graph visualizations to present information, like Ariadne [42] and Tesseract [35], might be ineffective in large-scale projects because graphs can be hard to understand when they reach a couple dozens nodes and edges due to overlapping nodes and edges [27].

In order to address this limitation in the design of current collaborative tools, we conducted an empirical study to find out the required scalability of such tools. Our investigation used the approach proposed by Cataldo and colleagues [4] to automatically compute, based on historical information, the set of actors a developer needs to be aware of, when performing a software development task. A developer needs to be aware of the members of this set of actors because they are "the likely set of workers that a particular individual might need to communicate with." This set of actors is called the *coordination requirements* (CRs) of an actor. By computing the CRs of software developers involved in a large-scale project, we were able to better assess the scalability that a collaborative tool must provide to effectively facilitate the coordination of software development tasks.

Therefore, this paper presents the results of an empirical study conducted using data from nearly 11 months of development of a large-scale distributed project, the IBM Rational Team Concert™ (RTC) project [17]. We computed the coordination requirements of the developers involved in this project to understand how large these coordination requirements were, as well as their composition regarding developers' team membership and geographical location.

In contrast to our previous work [8, 9] in which the identification of coordination requirements was performed in monthly intervals, in this paper we computed the coordination requirements in four different time-frames: 1-day, 7-day, 15-day, and 30-day intervals. Our goal with this analysis is to more clearly understand the scalability of the coordination requirements in both short- and long-term development activities to encompass the different types of tasks performed by software developers. Our results suggest that even using different time-frames, current collaborative tools are still not suitable for handling the scalability of software developers' coordination requirements in large-scale projects.

The rest of this paper is organized as follows. Section 2 presents the background and motivation for our work. Section 3 describes the methodology of the study, describing the scenario and how the coordination requirements were computed. The next section presents the study results, with the information about the computed coordination requirements, their descriptions, and analysis. Section 5 presents the discussion of our results and their implications. Finally, in Sect. 6, we present our conclusions and ideas for future work.

2 Background and motivation

Costa [7] conducted a literature review of several collaborative tools and uncovered that collaborative software development tools present relevant information to their users in one of three different ways: (i) using graphs integrated into the development environment, like Ariadne [42] and LightHouse [11]; (ii) using "annotations" that are integrated into the development environment. In this case, the "typical" approach is to present decorators in the Eclipse environment like in Palantír [36] and Jazz [6]; and (iii) as independent tools, i.e., using visualizations out of the development environment. A seminal example of this situation is Elvin's ticket tape approach [14, 15]. Another example is Tesseract [35].

A common problem across these tools is that they were often designed for small teams (defined in the literature as groups of 5 to 15 individuals [31]), therefore, it is not possible to ensure that they can handle large amounts of information [38]. For instance, tools that use graph visualizations can be ineffective in large-scale projects, since graphs can be hard to understand when they reach a couple hundreds nodes and edges due to overlapping nodes and edges [27]. Furthermore, scalability is an important aspect to consider during the design of collaborative tools [8]. Consider, for instance, the following scenario: a developer from Montreal (Canada) communicates with contributors from Los Angeles (US) and Montreal while working in a particular task. If we take into account only the number of locations (two in this case), this seems to be a relatively simple task compared with other contexts with five or more distributed locations on different continents. However, depending on the number of people that the developer needs to communicate with to perform a task, this task might become much more complex. A situation where he needs to communicate with two people from Montreal and only one from Los Angeles is much simpler than if one needs to be in contact with 15 people from Montreal and 12 from Los Angeles. In short, it is important to understand the number of people from other teams and locations that every developer needs to communicate/coordinate her work with, since the greater the need for communication and coordination with developers from different teams and locations, the more difficult is to align developers' activities.

Thus, a question arises: given a software developer, how many developers from other teams and locations this developer needs to coordinate her work with during the course of the project? To answer this question, it is necessary to identify the likely set of developers that a particular software developer might need to communicate and coordinate with for a given development task. Following our previous work [4, 8], we call this set as the coordination requirements (CRs for short) of a software developer.

In a recent study, de Souza and Redmiles [13] identified work practices used by software developers to identify their "awareness network," i.e., the set of actors that a given software developer needed to monitor and display their actions to [39] in order to successfully get their work done. Cataldo's approach to compute coordination requirements is a method for automatically identifying this network. As such, this method can be used to augment collaborative and awareness tools [22, 23]. This was an important reason that led us to select this approach. In addition, previous research has shown that software development productivity [3, 4] and software quality [2] improve when the identified coordination requirements are matched with appropriate coordination activities. Additional work has shown that the coordination requirements, as computed by Cataldo and colleagues, can be used in the construction of recommendation systems to facilitate the identification of experts in parts of the code [33] and tools for assessing coordination problems [35].

In addition to identifying the coordination requirements of all the developers in the RTC project, we analyze these CRs with respect to their scale (size) and composition (team membership and location). This is done to inform the design of collaborative software development tools. This aspect, and all other aspects of our methodology to study the scale of the coordination requirements and their composition, is presented in the next section.

3 Methodology

3.1 Study scenario

Recently, IBM made available the data from the software repositories of the project that developed the IBM® Rational® Jazz™ (RTC) collaboration platform.[1] RTC is a software development environment that integrates development, collaboration and project management features in the same platform [17]. A developer can, for example, receive notifications about events, view artifacts that other developers are changing, identify which developers are currently working among other functions. More specifically, IBM made available the data generated during the development of the first version of the product. By making this data available, IBM offered a unique opportunity for researchers to study collaborative activities of a real software development team [17]. In fact, many papers have been published with analysis of the RTC dataset.

We conducted an empirical study using data from the RTC project. The data available contained information about a period of 11 months (June 2007 to May 2008), corresponding to the development of the pre-releases 0.5 and 0.6. All RTC artifacts are stored in the same repository and linked together according to the relationships among the development activities. For example, a comment is associated with a task, called *Work Item*, which in turn is associated with a set of source files modified, called *Change Sets*. The integration between the different artifacts allows a researcher to study real collaborative software development activities, since she does not need to create her own heuristics to associate these artifacts [44].

A total of 161 developers worked in the project during the period analyzed and they were grouped in 33 teams distributed in 21 different locations across US, Canada, and Europe. Project members belonged to one or more functional teams, each one having its own manager. These teams reported their activities for the project management committee, which is composed by a group of people who are responsible for the overall coordination of the project.

Our analyses focused on the data from release 0.6 which represented 86.6 % of all development tasks and the work of the 17 core development teams. Thus, our analysis is based on the dataset from 77 members of the project who were involved in 23,359 tasks associated with 37,323 change sets.

3.2 Computing coordination requirements

Coordination requirements for each software developer from the RTC project were computed using the approach proposed by Cataldo and colleagues [3] and are outlined in the following paragraphs.

Step 1: Data collection

All the relevant data of the RTC project was stored in a single database. Therefore, we implemented several queries to extract the following data:

- *Modification Requests (MRs)*: represent the development tasks performed in a project. A task may consist of fixing a defect, implementing a new product feature or improving existing functionality;
- *Commits*: represent the sets of source code files modified by a developer as part of working on a MR;
- *Contributors' teams*: represent the formal teams that each developer belongs to in the project. Most of the RTC members are part of more than one formal team. These formal teams can represent traditional development process steps such as requirements, development, or maintenance. However, in other cases, formal teams represent responsibilities of a particular component of the software system such as, Web UI, memory management or map layout; and

[1] IBM, the IBM logo, ibm.com, and Rational are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

- *Geographical locations of contributors*: the cities where each project contributor worked. This information was extracted from Human Resources records since the RTC repository does not contain this type of information. We used this information to determine the locations involved in a particular project.

Step 2: Identification of Outliers

In the RTC project, we found modification requests (MRs) with very long resolution times. Those MRs tended to be enhancements with very low priority levels. In order to avoid bias in our results by this peculiar type of tasks, we decided to consider only the subset of MRs that had a maximum resolution time of the mean resolution time plus one standard deviation. A similar analysis was used for the commits: we only considered those MRs associated to commits that modified a maximum number of files that is less than the mean number of modified files plus 1 standard deviation. The rationale behind eliminating these particular commits is related to the fact that those commits represent special situations such as a software license being changed or a major refactoring that was not representative of a typical development activity [28].

Step 3: Group Commits in different time intervals

The first step in the process of identifying the coordination requirements of a given developer is to group the commits into representative units of time. In our own previous work [8], we used 1-month time intervals.[2] In fact, as we discussed in [8], the period used for the analysis is a choice of the researcher, not a requirement from the approach: different authors use different time-frames. In this work, we wanted to explore different time-frames and understand how they could impact the coordination requirements, and the associated scalability of collaborative tools supporting large-scale projects. Therefore, we analyzed short (1 day), medium (7 and 15 days), and long-term (30 days) time intervals in order to encompass the different types of software development tasks performed by developers. In other words, while the analysis of 30-day and 15-day intervals provides an overall assessment of the coordination needs of software developers, the analyses of 7-days and 1-day intervals provide a richer description of the daily work of software developers. In particular, Gersick [18, 19] suggests that important transitions occur halfway during tasks, and since 30-day periods included all the work associated with more than 90 % of all modification requests across all five projects in [8], we decided to compute 15-day intervals as well. We computed the average number of developers for each time interval and observed that the development activities occur continuously with an average of 28.66 developers per day (standard deviation of 17.47), 54.9 developers per week (s.d.: 12), 62.69 per 15-day intervals (s.d.: 8.47), and 67.6 developers per month (s.d.: 5.8).

Step 4: Calculating the Coordination Requirements

In this step, since we have all the data necessary, we can effectively compute the coordination requirements. Cataldo's technique involves constructing two matrices. The *Task Assignments* matrix is a Person by File matrix that indicates which source code files each developer changed as part of a development task. The *Task Dependencies* matrix captures the logical dependencies that exist between the various source code files that constitute a software system. Then the coordination requirements are computed as a matrix with the following expression: Task Assignments X Task Dependencies X transpose (Task Assignments). The resulting coordination requirements matrix is a People X People matrix where each cell outside the diagonal indicates the need for coordination of the person i with the person j. As discussed in the previous step, in our analyses, the matrices above mentioned contained data aggregated at four different time intervals.

Step 5: Computing the various measures of interest

Using the coordination requirements matrices described above, we computed a number of measures to understand the needed scalability of collaborative tools. Initially, we computed the descriptive statistics for the entire 11-month development period according to each time intervals, i.e., we computed the average number of coordination requirements, the maximum and minimum CRs and other typical descriptive statistics for 1, 7, 15, and 30-day intervals. These results are presented in Sect. 4.1. The second set of metrics was computed for each time interval instead of the entire development period. In other words, we computed the average number of software developers that each developer needs to be aware of, considering the developers' teams and locations for each 1, 7, 15, and 30-day intervals. In this case, we divided the metrics into 2 different aspects: first, for developers who are on the same team vs. different teams, and second, for developers who are on the same location vs. different locations. Results from these analyses are presented in Sects. 4.2 and 4.3.

4 Results

4.1 Initial Results

Table 1 below presents the first set of descriptive statistics considering the average number of coordination require-

[2] Cataldo et al. [4]. Used 1-week intervals solely for their analysis of the how the coordination requirements changed in a weekly basis.

ments per software developer. For the sake of completeness, we are reporting averages, skewness and other values, but we are interested in maximum and minimum values because of their impact in the scalability of collaborative tools.

The descriptive statistics were computed considering the coordination requirements of all developers for the entire 11-month development period. For instance, the mean number of CRs for 30 days was computed as follows. First, for each 30-day interval, we calculated the CRs for each developer. As a result, we had the coordination requirements of 77 developers in 11 different 30-day intervals, i.e., a 77×11 matrix of CRs. Second, we computed the descriptive statistics based on the data available in this matrix. By doing that, we were able to find the maximum number of coordination requirements that existed for any given developer in all 30-day intervals. A similar procedure was used to compute the descriptive statistics in the other time intervals.

Note also that this approach is different from what we describe in [8] and [9], where we reported the maximum *average* value (i.e., the maximum value found among the 11 averages calculated for each 30-day interval). In this paper, we are reporting the *actual* maximum coordination requirements.

Looking at the values in Table 1 it is possible to observe that in the four different time intervals considered in our analysis there were developers who had zero coordination requirements (the minimal values). In other words, some of the developers in the RTC project had no need to coordinate their work with other developers during these periods. On the other hand, the maximum values for each interval were 25 (1-day), 52 (7-day), 56 (15-day), and 63 (30-day). The collaborative tools reported in the literature [7] would have problems handling these values because they were not designed to be scalable. For instance, Ariadne [42] represents developers as nodes in a graph. In this case, the graph would contain about between 25 and 63 different nodes, which would make it hard to understand. As we discussed in Sect. 2, some of these tools use graph-based representations that would become problematic with such large values.

Furthermore, the difference between the minimum and maximum values suggests that in the RTC project there were situations where some developer needed to coordinate their activities with *few* other developers, while in other situations other developers needed to coordinate their work with *several* other developers. In other words, a coordination tool should be able to support both contexts: one where the coordination requirements are small and other where the coordination requirements are very large.

Table 1 Descriptive statistics considering the average number of CRs

	1 day	7 days	15 days	30 days
Mean	0.60	8.64	19.16	30.77
Median	0	0	10	40
Std. dev.	2.29	12.92	19.81	23.06
Max. value	25	52	56	63
Min. value	0	0	0	0
Skewness	4.91	1.32	0.38	−0.29
Kurtosis	26.97	0.51	−1.49	−1.53

4.2 Analysis of the coordination requirements according to team membership

Table 2 summarizes the descriptive statistics of the coordination requirements considering the different time intervals analyzed. Again, our focus is on the maximum and minimum values reported in the Table. The smallest value in this case is 16 for different teams in the 1-day intervals. This means that in one day during the RTC project, a software developer needed to potentially coordinate his work with 16 other software developers working on teams different from his own. In contrast, another developer in the RTC project needed to coordinate his work with 57 software developers from different teams. Finally, the minimal values are zero, again suggesting that some developers are in a more favorable situation regarding their effort to coordinate their work.

Table 2 Descriptive statistics considering the average number of CRs per developer in the context of team membership

	1 day		7 days		15 days		30 days	
	Same team	Different teams	Same team	Different teams	Same team	Different teams	Same team	Different teams
Mean	1.84	0.37	10.96	3.25	20.32	6.77	28.79	10.67
Median	0	0	6	1	21	4	34	7
Std. dev.	3.43	1.25	11.61	5.77	15.80	9.19	17.17	11.91
Max. value	21	16	44	38	52	49	57	57
Min. value	0	0	0	0	0	0	0	0
Skewness	2.39	6.12	0.84	3.05	0.12	2.23	−0.38	1.88
Kurtosis	5.62	48.74	−0.43	10.53	−1.45	4.78	−1.25	2.89

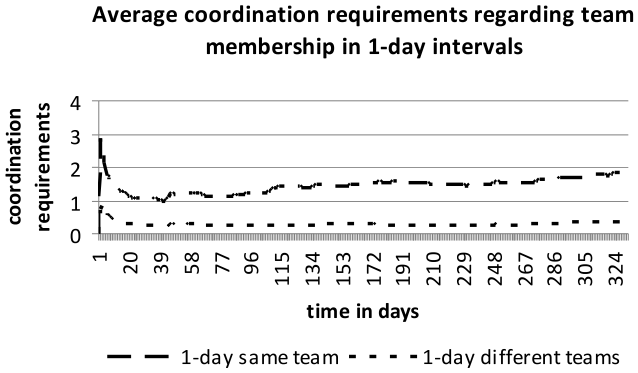

Fig. 1 Average number of coordination requirements per developer in the same (different) team(s) considering the 1-day time interval

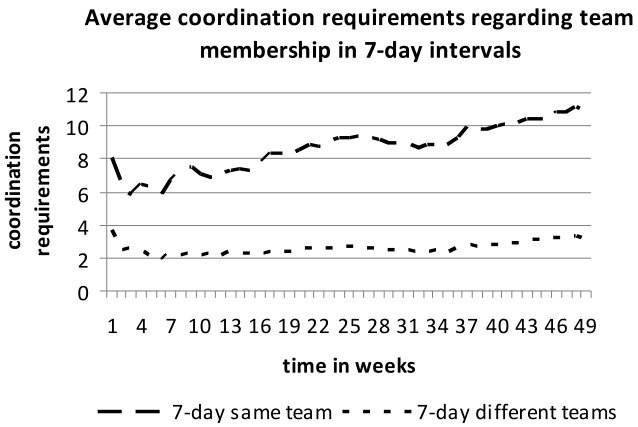

Fig. 2 Average number of coordination requirements per developer in the same (different) team(s) considering the 7-day time interval

Figures 1, 2, 3 and 4 represent graphically the evolution of the coordination requirements plotting two values: the average number of coordination requirements (i) for the same theam and (ii) for different teams. In these figures, the X-axis represents the time period of analysis, while the Y-axis is used to represent the average number of coordination requirements per developer, i.e., the average number of other developers that every developer should be communicating/coordinating his work with. The results indicate that for each time interval, the average number of coordination requirements for the same team is higher than the average number of coordination requirements for different teams. That is expected, since in general, organizational theory principles [40, 41] suggest that activities tend to be assigned to developers in the same team so that people have more interactions with these developers. This is important because the coordination between actors from different teams is generally more difficult than between members of the same team, including in software development teams [12, 30].

Figures 1–4 also indicate that while the curves that represent different teams have a small variation between their

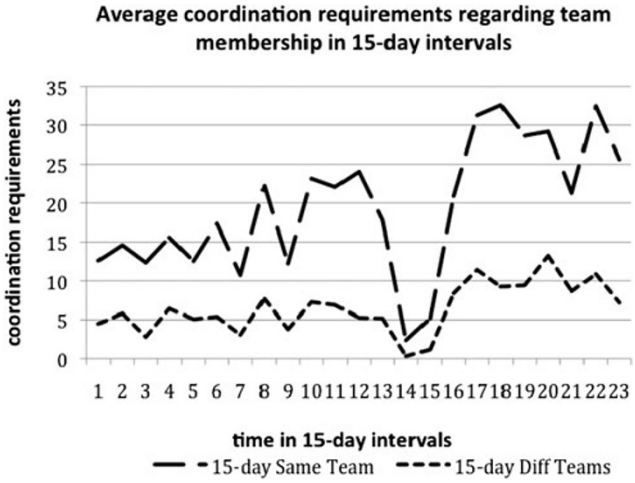

Fig. 3 Average number of coordination requirements per developer in the same (different) team(s) considering the 15-day time interval

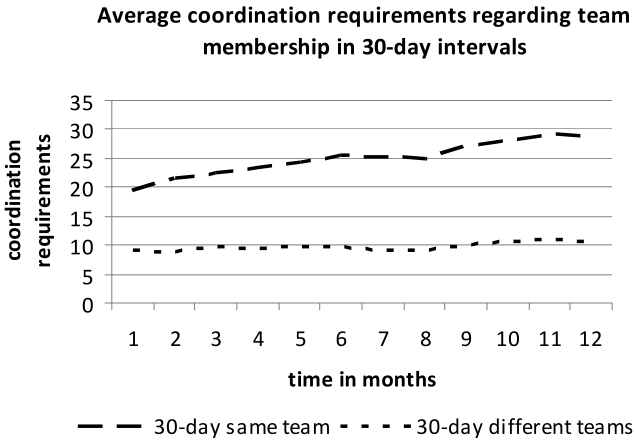

Fig. 4 Average number of coordination requirements per developer in the same (different) team(s) considering the 30-day time interval

maximum and minimum values, the same team situation has a larger variation. This suggests that, on average, of the overall set of coordination requirements from any given developer, the number of developers who belong to different teams remains more or less constant, while the number of developers from the same team changes more often.

4.3 Analysis of the coordination requirements according to geographical locations

This section presents our analysis of the coordination requirements involving developers' geographical locations. As discussed in Sect. 3.1, the Rational Team Concert project involves software developers spread in 21 different locations across US, Canada, and Europe.

We start by reporting the descriptive statistics for the entire development period, and according to the different time intervals. This is presented on Table 3. Since we are interested in understanding the scalability of collaborative tools,

Table 3 Descriptive statistics considering the average number of CRs per developer by location

	1 day		7 days		15 days		30 days	
	Same location	Different locations	Same location	Different locations	Same location	Different locations	Same location	Different locations
Mean	0.65	1.56	2.87	11.34	4.89	22.21	6.53	32.93
Median	0	0	2	7	5	25	7	37
Std. dev.	1.23	3.16	2.71	11.89	3.41	15.87	3.51	15.93
Max. value	9	22	13	49	13	52	14	61
Min. value	0	0	0	0	0	0	0	0
Skewness	2.62	2.45	0.66	0.68	0	−0.16	−0.32	−0.87
Kurtosis	8.33	6.07	−0.62	−0.67	−1.26	−1.39	−0.68	−0.26

we again focus on the maximum coordination requirements values obtained. In this case, they range from 9 (same location, 1-day interval) to 61 (different locations, 30-day interval). While 9 is a value that can potentially be supported by collaborative tools, 61 cannot (based on our literature review described in [7]), which again suggests the need to design collaborative tools for scalability.

In contrast to the overall picture presented in Table 3 above, Figs. 5, 6, 7 and 8 detail the average coordination requirements in each interval of analysis plotting the values according to the same location or different locations situation.

Similarly to the teams context, in situations involving many geographical locations the standard recommendation is to arrange the developers in such a way that people from different locations do not have to interact as much as developers in the same location [20, 25]. This is recommended because coordination across distance is more difficult due to the lack of informal communication that is necessary for effective coordination [26, 29].

Nevertheless, Figs. 5, 6, and 8 show a different scenario: the average number of coordination requirements between different locations is generally higher than in the same location. In other words, in general, a software developer of the RTC project needs to be aware of activities from a higher number of developers who are physically located in other sites compared to collocated developers. In fact, observing Figs. 5–8 is possible to observe that the "same location" line keeps relatively constant as the project unfolds, while the average number of coordination requirements from different locations not only presents a higher variation, but increases over time.[3] A possible explanation is that, as the project progresses toward the end of the milestone, developers need to integrate their software components, which require them to engage in more direct communication with their colleagues.

[3]The exception in this case is Fig. 7 for the 15-day interval because of its decrease toward zero around the 14th or 15th 15-day interval.

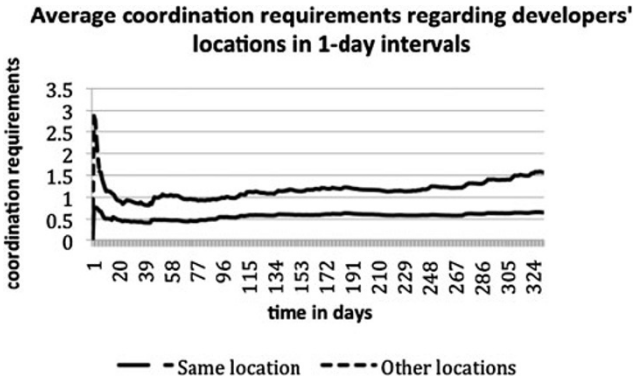

Fig. 5 Average number of CRs per developer in 1-day intervals according to developers' locations

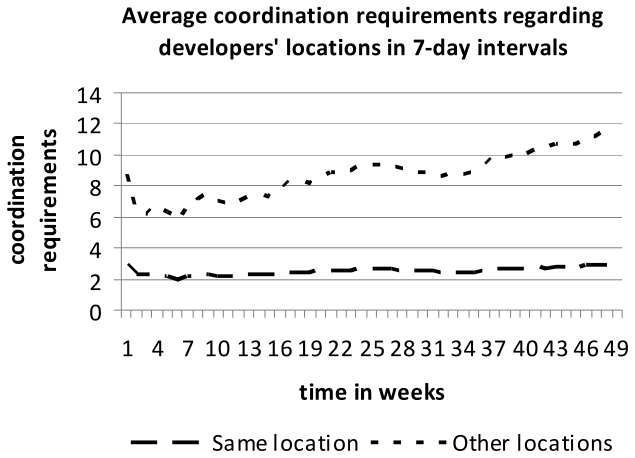

Fig. 6 Average number of CRs per developer in 7-day intervals according to developers' locations

5 Discussion

In this paper, we examine the scale and temporal evolution of coordination requirements in the Rational Team Concert (RTC) project, a large project that involves 161 developers, 33 development teams, and 21 locations across US, Canada, and Europe. More specifically, our analyses focused on the

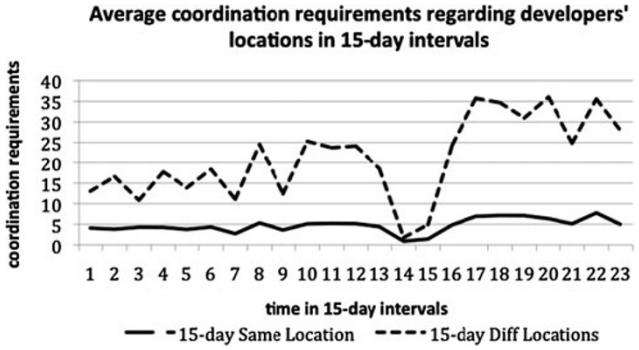

Fig. 7 Average number of CRs per developer in 15-day time intervals according to developers' locations

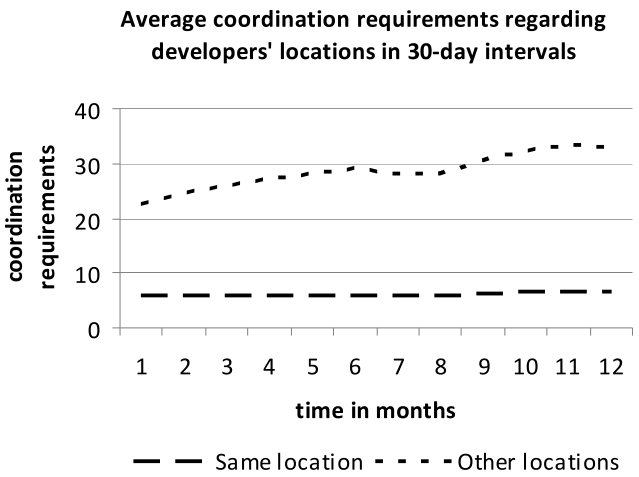

Fig. 8 Average number of CRs per developer in 30-day intervals according to developers' locations

data from pre-release 0.6, which represented 86.6 % of all development tasks and on the work of the 17 core development teams and 77 developers.

We analyzed this project considering four different time intervals: 1 day, 7 days, 15 days, and 30 days. We considered these time frames because they were used in past work [4, 8] and also because we wanted to cover different types of activities performed by software developers, and consequently with different durations. In addition, Gersick's work [18, 19] suggests that important changes in group dynamics take place around the half of the time allocated to the project, therefore, we also considered 15-day intervals, since most modification requests lasted about a month [8].

By carefully inspecting the maximum values reported in Tables 1, 2, and 3, it is clear that even in different timeframes current collaborative tools will face problems in large-scale projects because they need to handle information from up to 63 different software developers that belong to different teams or that are in different locations. If we consider that the total number of developers in this project was 77, this basically meant that some developers needed to interact with almost all ($63/77 \approx 82$ %) other developers in the project. In fact, most collaborative tools are designed solely for a small group of developers [38], a situation very different from what we identified in the RTC project. For instance, tools like Palantír [36] and RaisAware [10] present information from developers as decorators in the Eclipse IDE, which means they would have to present information from up to 60 different software developers into a single Eclipse IDE.

As for the team membership aspect, we found relatively low values for the median and mean CRs in the different time frames (see Table 2). This is expected since organizational theory suggests that team members should engage in more direct communication with people in their same team [40, 41]. However, by looking at maximum values from Table 2, it is possible to notice large values for the different team situations, namely 16 (1-day), 38 (7-day), 49 (15-day), and 57 (30-day). Coordinating across organizational boundaries is generally more difficult than within the same team because of the lack of context, shared goals and opportunities for informal interaction that help to build interpersonal relationships [12, 30]. This difficulty combined with large coordination requirements means that is even more important to properly design collaborative tools, because they have the potential to simplify this situation and facilitate the coordination of work.

Previous work [26, 29] suggests that coordination across geographical boundaries is more difficult when compared to coordination in the same site. This occurs because when actors are in the same location they can use physical cues from their environment to figure out when to interrupt their colleagues without disrupting the flow of work, they have more opportunities for informal interactions, and these informal interactions are very important in the coordination of daily work. Despite that, our results regarding the locations of software developers are unexpected: the maximum values for the different location condition is higher than for the same location in the four different time-frames. Again, this illustrates the potential of collaborative tools to facilitate this process.

In general, current collaborative tools are typically designed for small teams and are not able to handle large amounts of information [38]. In addition, they fail to provide a global understanding of interactions among teams, what makes them ill-suited to software development projects that involves multisites [38], a more and more common scenario nowadays like the RTC. For example, Palantír is a tool that was originally developed using decorators to present developers' information to its users. This approach is useful to grasp the user's attention, but it does not present relevant information such as: Is the developer located in the same location than me? Is he a member of my development team? We have shown in our previous work that these questions

are relevant [8]. Other examples of tools whose visualizations were not designed to be scalable are the ones that use (social)-graphs like Ariadne [42] and Tesseract [35].

Our results also suggest that collaborative tools need to represent information about the team membership and location of the other collaborators. Or, at least, to be integrated with other tools that have this information available so that it is easy to gather out this information from the collaborative tool. Location is particularly important because if people are in different time-zones, this means that their type of interaction, synchronous or asynchronous, has to be different. In the RTC project, as our results suggest, developers need to coordinate their work fairly often with developers from other locations.

In short, our results suggest that coordination and awareness tools should be designed for scalability. Furthermore, collaborative tools should be able to dynamically adjust the visualization according to the number of coordination requirements, teams and locations involved. If the coordination requirements of the developers are low, then representations like graphs or a tickertape [14, 15] would be adequate. However, if the coordination requirements are high, a different type of visualization would be necessary, for instance with dashboards [43] including information about the relationships between teams and locations.

Finally, it is also worth noticing how the coordination requirements vary among the software developers from the RTC project: while some developers are potentially overwhelmed coordinating their work with several other developers, others are in a more favorable situation. This finding was consistent in all different evaluated timeframes and also across teams and locations. The implications of this result are particularly relevant for the design of collaborative tools because most of them only support one type of visualization, i.e., they do not take into account the variability of the coordination requirements among the different social actors. Furthermore, this large difference between the coordination requirements of developers raises interesting research questions, including but not limited to:

- Who are the developers with the highest (or lowest) coordination requirements?
- Why there is such a difference in their coordination needs?
- Are higher coordination needs correlated with particular roles, productivity, expertise, or any other factor?
- And, finally, in the context of collaborative tool design, given this different in coordination requirements, should these different "types" of developers use the same collaborative tools, or should they use different tools?

We plan to explore these questions in our future work.

6 Conclusions and future work

In this paper, we examine the scale of coordination requirements in the Rational Team Concert (RTC) project, a large-scale distributed software development with developers spread around the globe in sites in the US, Canada, and Europe. We computed the coordination requirements [4] of all the developers for different time-frames: 1-day, 7-day, 15-day, and 30-day. By doing that, we argue that we are able to cover different types of tasks performed by software developers during their activities. This is, in fact, the main difference of the work reported in this paper and our previous ones [8, 9].

As the Rational Team Concert is a large project, we expected high values for the coordination requirements of the developers, and indeed in many cases this was true, but in other situations the values found were low. This suggests an imbalance in the coordination effort performed by the developers in the project [8].

Another result of our study is that while the coordination requirements between developers of the same team tend to be higher than the coordination requirements between people of different teams (as expected), the coordination requirements between people of different locations are generally higher than the ones involving people of the same location. That is a surprising result, since past work suggests that the coordination of software development activities is more difficult when the developers are geographically distributed [20, 25]. We plan to explore this aspect further in our future work.

In general, our main contribution in this paper is to draw attention of collaborative tool designers to the need to design for scalability, i.e., to design collaborative tools that are able to properly present large amounts of information from different social actors. In addition, filtering mechanisms might be necessary to allow an actor to easily view this information and make informed decisions about, when and to which extent, he needs to coordinate his work with his colleagues.

Acknowledgements The first and second authors were supported by the Brazilian Government under grant CNPq 473220/2008-3 and by the Fundação de Amparo à Pesquisa do Estado do Pará through "Edital Universal N°003/2008." The third author is grateful for the financial support provided by Accenture, Robert Bosch, and IBM that made this research possible.

References

1. Bowers J (1994) The work to make the network work: studying CSCW in action. In: Conference on computer-supported cooperative work, Chapel Hill, NC, USA. ACM, New York
2. Cataldo M, Herbsleb J (2010) Coordination breakdowns and their impact on development productivity and software failures. Pittsburgh, Institute for Software Research, Carnegie-Mellon University. ISR-10-104

3. Cataldo M, Herbsleb JD et al (2008) Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In: Proceedings of the second ACM-IEEE international symposium on empirical software engineering and measurement, Kaiserslautern, Germany. ACM, New York
4. Cataldo M, Wagstrom PA et al (2006) Identification of coordination requirements: implications for the design of collaboration and awareness tools. In: 20th conference on computer supported cooperative work, Banff, Alberta, Canada. ACM, New York
5. Cheng L-T, Hupfer S et al (2003) Jazzing up eclipse with collaborative tools. In: OOPSLA workshop on eclipse technology exchange, Anaheim, CA, USA
6. Cheng L-T, Hupfer S et al (2003) Jazz: a collaborative application development environment. In: ACM SIGPLAN conference on object oriented programming systems languages and applications, Anaheim, CA, USA. ACM, New York
7. Costa JMdR (2011) Escalabilidade e Evolução das Redes de Awareness em um Grande Projeto de Desenvolvimento Distribuído de Software. MSc, Universidade Federal do Pará
8. Costa JMdR, Cataldo M et al (2011) The scale and evolution of coordination needs in large-scale distributed projects: implications for the future generation of collaborative tools. In: ACM conference on human factors in computing systems, Vancouver, Canada, pp 3151–3160
9. Costa JMdR, De Souza CRB (2010) Analyzing the scalability of awareness networks in a distributed software development project. In: Brazilian symposium on collaborative systems
10. Costa JMdR, Feitosa RM et al (2009) RaisAware: uma ferramenta de auxiílio á engenharia de software colaborativa baseada em anaálises de dependências. Scientia 9(2):7–36
11. da Silva IA, Chen P et al (2006) Lighthouse: coordination through emerging design. In: OOPSLA eclipse technology exchange workshop, pp 11–15
12. de Souza CRB, Redmiles D (2009) On the roles of APIs in the coordination of collaborative software development. Comput Support Coop Work 18(5–6):445–475
13. de Souza CRB, Redmiles D (2011) The awareness network, to whom should I display my actions? And, whose actions should I monitor? (forthcoming). IEEE Trans Softw Eng 1–18
14. Fitzpatrick G, Kaplan S et al (2002) Supporting public availability and accessibility with Elvin: experiences and reflections. Journal of Computer Supported Cooperative Work 11(3–4):299–316
15. Fitzpatrick G, Mansfield T et al (1999) Augmenting the workaday world with Elvin. In: 6th European conference on computer supported cooperative work, Copenhagen, Denmark. Kluwer Academic, Norwell
16. Fitzpatrick G, Marshall P et al (2006) CVS integration with notification and chat: lightweight software team collaboration. In: Proceedings of the 2006 20th anniversary conference on computer supported cooperative work, Banff, Alberta, Canada. ACM, New York
17. Frost R (2007) Jazz and the eclipse way of collaboration. IEEE Softw 24:114–117
18. Gersick CJG (1988) Time and transition in work teams: towards a new model of group development. Acad Manage J 31(1):9–41
19. Gersick CJG (1989) Making time: predictable transitions in task groups. Acad Manage J 32(2):274–309
20. Grinter R, Herbsleb J et al (1999) The geography of coordination: dealing with distance in R&D work. In: ACM conference on supporting group work (GROUP '99), Phoenix, AZ. ACM, New York
21. Grinter RE (2003) Recomposition: coordinating a web of software dependencies. Journal of Computer Supported Cooperative Work 12(3):297–327
22. Gutwin C, Greenberg S (1996) Workspace awareness for groupware. In: Conference companion on human factors in computing systems: common ground, Vancouver, British Columbia, Canada ACM, New York, pp. 208–209
23. Gutwin C, Greenberg S (2002) A Descriptive framework of workspace awareness for real-time groupware. Journal of Computer Supported Cooperative Work 11(3–4):411–466
24. Herbsleb JD, Atkins DL et al (2002) Introducing instant messaging and chat in the workplace. In ACM conference on human factors and computing systems (CHI'2002), Minneapolis, Minnesota USA. ACM, New York
25. Herbsleb JD, Grinter RE (1999). Architectures, coordination, and distance: Conway's law and beyond. IEEE Softw 63–70
26. Herbsleb JD, Mockus A et al (2001) An empirical study of global software development: distance and speed. In: International conference on software engineering, Toronto, Canada, IEEE, New York
27. Herman I, Melancon G et al (2000) Graph visualization and navigation in information visualization: a survey. IEEE Trans Vis Comput Graph 6(1):24–43
28. Hindle A, German DM et al (2008) What do large commits tell us?: a taxonomical study of large commits. In: Proceedings of the 2008 international working conference on mining software repositories, Leipzig, Germany. ACM, New York, pp 99–108
29. Kraut R, Egido C et al (1990) Patterns of contact and communication in scientific research collaborations. In: Galegher, J., Egido, C., Kraut, R., Intellectual teamwork: social and technological foundations of cooperative work. Erlbaum, Hillsdale, pp 149–172
30. Kraut RE, Streeter LA (1995) Coordination in software development. Commun ACM 38(3):69–81
31. Levine JM, Moreland RL (1990) Progress in small group research. Annu Rev Psychol 41(1):585–634
32. McDonald DW, Ackerman MS (2000) Expertise recommender: a flexible recommendation system and architecture. In: Conference on computer supported cooperative WORK (CSCW '00), Philadelphia, PA
33. Minto S, Murphy G (2007) Recommending emergent teams. In: Proceedings of the fourth international workshop on mining software repositories. IEEE Comput Soc, Los Alamitos, p 5
34. Sarma A, Bortis G et al (2007) Towards supporting awareness of indirect conflicts across software configuration management workspaces. In: Proceedings of the twenty-second IEEE/ACM international conference on automated software engineering, Atlanta, Georgia, USA. ACM, New York
35. Sarma A, Maccherone L et al (2009) Tesseract: interactive visual exploration of socio-technical relationships in software development. In: Proceedings of the 31st international conference on software engineering, IEEE Comput Soc, Los Alamitos, pp 23–33
36. Sarma A, Noroozi Z et al (2003) Palantír: raising awareness among configuration management workspaces. In: Twenty-fifth international conference on software engineering, Portland, Oregon
37. Sarma A, Redmiles D et al (2008) Empirical evidence of the benefits of workspace awareness in software configuration management. In: Proceedings of the 16th ACM SIGSOFT international symposium on foundations of software engineering, Atlanta, Georgia. ACM, New York, pp 113–123
38. Sarma A, van der Hoek A (2006) Towards awareness in the large. In: International conference on global software engineering, Florianopolis, Brazil. IEEE Press, New York
39. Schmidt K (2002) The problem with 'awareness'—introductory remarks on 'awareness in CSCW'. Journal of Computer Supported Cooperative Work 11(3–4):285–298
40. Scott WR (2003) Organizations: rational, natural, and open systems. Prentice Hall, Upper Saddle River
41. Thompson JD (1967) Organizations in action: social sciences of administrative theory. Transaction Publishers, New Brunswick

42. Trainer E, Quirk S et al (2005) Bridging the gap between technical and social dependencies with Ariadne. In: Eclipse technology exchange, San Diego, CA
43. Treude C, Storey M-A (2010) Awareness 2.0: staying aware of projects, developers and tasks using dashboards and feeds. In: Proceedings of the 32nd ACM/IEEE international conference on software engineering, vol 1, Cape Town, South Africa. ACM, New York, pp 365–374
44. Wolf T, Nguyen T et al (2008) Does distance still matter? Softw Process Improv Pract 13(6):493–510