POLYPUBLIE

Are we working well with others? How the multi team systems impact software quality

Mathieu Lavallée, Pierre N. Robillard

Document en libre accès dans PolyPublie
Open Access document in PolyPublie

Date	2018
Type	Article de revue / Article
Version	Version officielle de l'éditeur / Published version
URL de PolyPublie PolyPublie URL	https://publications.polymtl.ca/3566/
Évaluation par les pairs Peer reviewed	Révisé par les pairs / Refereed
Conditions d'utilisation Terms of use	Creative Commons Attribution 4.0 International (CC BY)
Référence Citation	Lavallée, M., & Robillard, P. N. (2018). Are we working well with others? How the multi team systems impact software quality. e-Informatica Software Engineering Journal, 12(1), 117-131. https://doi.org/10.5277/e-inf180105

Publication officielle
Official publication

Titre de la revue Journal title	e-Informatica Software Engineering Journal (vol. 12, no. 1)
Maison d'édition Publisher	Wrocław University of Science and Technology
URL officielle Official URL	https://doi.org/10.5277/e-inf180105
Mention légale Legal notice	

Ce fichier a été téléchargé à partir de PolyPublie, le dépôt institutionnel de Polytechnique Montréal
This file has been downloaded from PolyPublie, the institutional repository of Polytechnique Montréal
https://publications.polymtl.ca

e-Informatica Software Engineering Journal, Volume 12, Issue 1, 2018, pages: 117–131, DOI 10.5277/e-Inf180105

Are We Working Well with Others? How the Multi Team Systems Impact Software Quality

Mathieu Lavallée*, Pierre N. Robillard*
*Département de génie informatique et génie logiciel, Polytechnique Montréal
mathieu.lavallee@polymtl.ca, pierre.robillard@polymtl.ca

Abstract

Background: There are many studies on software development teams, but few about the interactions between teams. Current findings suggest that these multi-team systems may have a significant impact on software development projects.
Aim: The objective of this exploratory study is to provide more evidence on multi-team systems in software engineering and identify challenges with a potential impact on software quality.
Method: A non-participatory approach was used to collect data on one development project within a large telecommunication organization. Verbal interactions between team members were analyzed using a coding scheme following the Grounded Theory approach.
Results: The results show that the interactions between teams are often technical in nature, outlining technical dependencies between departments, external providers, and even clients.
Conclusion: This article hypothesizes that managers of large software project should (1) identify external teams most likely to interfere with their development work, (2) appoint brokers to redirect external requests to the appropriate resource, and (3) ensure that there are opportunities to discuss technical issues at the multi-team level. Failure to do so could results in delays and the persistence of codebase-wide issues.

Keywords: multi team system, human interaction, quality management, team management, industrial study

1. Introduction

Five hundred years ago, John Donne wrote that "no man is an island". Individuals achieve great things by working together as a team. But many projects require more than an individual team to achieve success. "No team is an island" [1] would be a better description of modern project and organization management.

Teamwork has indeed long been identified as important to project success [2–4]. Teamwork in software development is no different, and software engineering research also highlighted the impacts that software development teams can have. As Watts S. Humphrey wrote, "Systems development is a team activity, and the effectiveness of the team largely determines the quality of the engineering" [5, p. 51]. Teams rarely work in isolation; teams are often interdependent of each other and must work together. Recent studies have shown the importance of these interactions between teams, whether on issues such as organization-wide knowledge sharing [6], coordination of multiple agile teams [7] or inter-team communication effectiveness [8].

This paper presents insights gained from the analysis of data collected in an exploratory study. These insights confirm the large amount of inter-team interactions, and identifies which teams were more closely connected to the development team. It also shows the role the developers play as middlemen between teams, for example between clients and testers. Finally, this study presents the importance of inter-team technical coordination, which is difficult if the organization

Table 1. Software engineering publications related to MTS in chronological order

Ref	Title (publication year)
[9]	Using open spaces to resolve cross team issue (2005)
[10]	Implementing Scrum in a distributed software development organization (2007)
[11]	Forming to performing: Transitioning large-scale project into agile (2008)
[12]	Fully distributed Scrum: Replicating local productivity and quality with offshore teams (2009)
[13]	Moving back to Scrum and scaling to Scrum of Scrums in less than one year (2011)
[14]	Scaling Scrum in a large distributed project (2011)
[15]	Scrum practice mitigation of global software development coordination challenges: A distinctive advantage? (2012)
[16]	Coordination in co-located agile software development projects (2012)
[17]	Practical Scrum-Scrum team: Way to produce successful and quality software (2013)
[18]	Coordination in large-scale agile software development: A multiteam systems perspective (2014)
[6]	Fostering effective inter-team knowledge sharing in agile software development (2015)
[19]	The effects of team backlog dependencies on agile multiteam systems: A graph theoretical approach (2015)
[20]	A multiple case study on the inter-group interaction speed in large, embedded software companies employing agile (2016)
[21]	The architect's role in community shepherding (2016)

only supports inter-team administrative coordination (i.e. resource planning and scheduling).

The next section presents the related work (Section 2), with a focus on the organizational psychology concept of multi-team systems and how it applies to software engineering. The methodology (Section 3) presents the context of the study and how the data was collected and analyzed. The results (Section 4) presents the data analysis, while the discussion (Section 5) presents our hypotheses and limitations to the conclusions of the study. The conclusion (Section 6) summarizes the hypotheses and presents future avenues of research. Note that this paper represents an extension of a previous shorter publication [22]. Some elements of the methodology were reused here, but the results and analyses are new.

2. Related work

The current software engineering literature uses different terms to define the interactions between multiple teams: inter-team, multi-team, cross-team, etc. However, these concepts are not always clearly defined, leaving the exact interpretation to the reader. The research field of organizational psychology has fortunately studied this topic extensively, regrouping them under the umbrella of multi-team systems, or MTS [23]. The MTS are defined as:

> Two or more teams that interface directly and interdependently in response to environmental contingencies toward the accomplishment of collective goals. MTS boundaries are defined by virtue of the fact that all teams within the system, while pursuing different proximal goals (e.g. writing a specific code module), share at least one common distal goal (e.g. creating a complete working software); and in so doing exhibit input, process, and outcome interdependence with at least one other team in the system [24].

Many studies have been published on single team dynamics in recent decades. Additionally, there is also a large body of knowledge on global or distributed software engineering, that is, multi-team systems spanning different sites across the globe. However, as far as we could find, there are few publications on the dynamics between co-localized teams. What should be done to make teams work together effectively within the same site at the organizational level?

> What is required for success in these kinds of MTSs is coordination both *within* and *between* teams [emphasis theirs]. That is, al-

> though interventions designed to create a system of strong, cohesive component teams may maximize performance at the team level, when ultimate system-level goals require synchronization between teams, more is needed. [...] MTS interventions must also address interdependencies between teams if performance across these kinds of complex systems is to be maximized [25].

Studies observing MTS in software engineering are still limited [18], with almost all studies found limited to Agile contexts and Scrum-of-Scrums meetings, as shown in Table 1.

Mike Cohn, an expert on the Scrum process, recommends a specific point in the agenda of "Scrum of Scrums" meeting, his version of MTS status meetings. Cohn recommends the addition of a question saying: "Are you about to put something in another team's way?" [26]. Cohn's recommendation outlines the importance of MTS and the impact one team can have on another. This recommendation was used in the field within "Scrum-of-Scrums" meetings, but with limited success [7]:

> Both case projects started using a model in which only one issue was discussed: impediments. However, this solution did not turn out well.[...] Both case projects still recognized the need for project-wide inter-team synchronization, but did not have any good solutions to the problem [7].

This shows that while the challenges of MTS projects are beginning to be better known, working solutions are still being tested [21].

2.1. Known challenges of MTS projects

This section presents a non-exhaustive list of challenges of MTS projects, based on what could be found in the literature. These three challenges were found to be most prevalent in the context of this study:

- Finding a compromise between team-level goals and MTS-level goals.
- Enabling effective communications and technical knowledge exchange at the MTS level,
- Planning the work at the MTS level.

One of the main MTS challenge is related to building a compromise between the objective of the local team goal and the overall goals of the MTS. In one software engineering case, the conflicting agendas of team members within different departments led to the failure of the project [13]. This challenge has a major impact on resource allocation. Organizational psychology researchers observed that "having to simultaneously work toward team-level goals along with MTS-level goals creates a demanding work environment" [25]. In software engineering, Santos et al., reached a similar conclusion. They studied knowledge sharing between teams in an Agile context [6]. They noted that the introduction of new MTS support practices requires more resources, which must be provided by the organization, otherwise the practice, and potentially the project, could fail.

Another challenge is the relative difficulty to ensure efficient communications at the MTS level, compared to communications within the team. A survey conducted by Kiani et al. noted that due to "lack of communication, almost fourth of respondents complained that work items they depended on have changed without any notification" [27]. Some basic Agile principles are also affected in MTS contexts. For example, face-to-face communications are easy at the team level, but are difficult to apply at the MTS level. It requires the organization to mix people from one team to another, which is not always possible [28]. "Boundary spanning", ensuring communications between the frontiers of the teams, is an important challenge within MTS [16, 20].

In the same vein, dissemination of technical information specific to a field of knowledge is also difficult. Local teams accumulate a significant amount of knowledge about the specific area in which they work. How can this knowledge be effectively communicated to the other teams in the MTS? If the project is particularly complex, it may also be difficult to get an overall view of the project [14]. Each team knows its own problems, which can be difficult to translate in a form understandable by other teams that might not have the same knowledge of the field.

A third challenge is related to how MTS coordination should be planned. Lanaj et al. found the following.

> Decentralized planning has positive effects on multiteam system performance, attributable to enhanced proactivity and aspiration levels. However, [...] the positive effects associated with decentralized planning are offset by the even stronger negative effects attributable to excessive risk seeking and coordination failures [29].

The study of MTS coordination has been identified by one study as "underdeveloped" [18]. However, MTS is a concept defined within the domain of organizational psychology. Research in software development already has a large body of knowledge pertaining to inter-team interactions within the domain of global and distributed software development [30]. While a global or distributed development team is a form of MTS, some MTS can be collocated in the same building. The team observed interacted with other teams which were almost all collocated within the same building. The context of this study is therefore different from the study of global and distributed software development, where the issues of geographical distance and temporal distance play a large role.

3. Methods

3.1. Industrial context

The study was performed on a large telecommunications organization with over forty years of experience in the industry. Throughout the years, the organization has developed a large codebase, which must be constantly updated. This study follows one such update project. The outcomes of this study are based on ten months of observation of a software development team involved in a two-year project for an internal client. The project involved a complete redesign of an existing software package used in the organization's internal business processes.

The technical challenge of this update project is that it requires the modification of COBOL legacy software, Web interfaces, mobile device integration and multiple databases. Its purpose is to manage work orders. To do this, it needs to extract data from multiple sources within the enterprise (employee list, equipment list, etc.) and send it to multiple databases (payroll, quality control, etc.).

The project was a second attempt to overhaul this complex package. A first attempt had been made between 2010 and 2012 but was abandoned after the fully integrated software did not work. Because this project was a second attempt, many specifications and design documents could be reused. Accordingly, the development followed a traditional waterfall process, as few problems were expected the second time around. This second attempt began in 2013 and was successfully deployed during October and November 2014.

The organization has no formal MTS coordination practices in place. Coordination at the MTS level is therefore mostly tacit. This means that when a team needs information from another team, a member of the first team has to directly contact another member of the second team. This causes some issues at the MTS level, because most developers in the team observed were new to the company [31] when the project started, and in some cases did not know who to contact in the other teams. Despite its tacit nature, an MTS exists. The need for coordination between the projects means that interactions between teams are required to perform the work.

This study observes a development team of nine members: one manager, four senior developers, two junior developers and two contract developers. The team was formed specifically for this project, of which seven are new to the organization (i.e. less than five years).

Note that the nature of this MTS is different from an MTS where several teams are working on the same project (e.g. a Scrum-of-Scrums development project). In the MTS observed, all teams had different projects, with their own goals and objectives. The development project studied was the responsibility of a single team, the team observed. However, to perform that project, that team could not do it alone, and had to seek help from other teams.

The objective of this study is to understand how a development team interacts with other external team to do its work. Therefore, the focus is on the development team. Who does the development team needs to talk to and why?

3.2. Study approach

The objective of the study was to identify the cause behind the introduction of quality problems during software development. Given the sensitive nature of problem identification within a large organization, it was decided to opt for a neutral approach. Data collection was to be performed using a non-participatory approach, to avoid organizational influence.

Data collection was limited to weekly status meetings because that is the avenue used by the organization to discuss and resolve MTS issues. Although there were certainly discussions between teams outside these weekly status meetings, the most important issues were discussed at these meetings.

A qualitative approach was chosen to better understand an area where many variables are not fully identified. The approach of this study uses the same rationale as Looney and Nissen:

> The present research is exploratory in nature, is not guided by extensive theory, and is approaching a "how" research question. Hence qualitative field research reflects an appropriate method [32].

3.3. Data collection methodology

This study is based on non-participant observation of the software development team's weekly status meetings. These meetings consisted of mandatory all-hands discussions for the eight developers assigned mostly full-time to the project, along with the project manager. These meetings included, as needed, developers from related external modules, testers, database administrators, security experts, quality control specialists, etc. The meetings involved up to 15 participants, and up to five additional participants through conference calls.

The team discussed the progress made during the previous week, the work planned for the coming week and obstacles to progress. The problems raised concerned resources and technical issues. Few decisions were taken at these meetings, the purpose being to share the content of the previous week's discussions between the different teams.

A round-table format was used, where each participant was asked to report on their activities. The discussions were open and everyone was encouraged to contribute. When a particular issue required too much time, participants were asked to set another meeting to discuss it. Meetings lasted about an hour.

The data presented in this study was collected over seven months during the last phase of the two-year project. It is based on 21 meetings held between January and July 2014. The same observer attended all the meetings and took note of who was involved in each interaction, the topic being discussed, and the outcome. A typical interaction would last between 5 and 30 seconds. The notes were then produced as quasi-verbatim transcripts.

3.4. Coding methodology

Due to the large amount of data collected, it is necessary to summarize the data obtained in order to find patterns. This summarization was performed using a coding methodology based on the grounded theory approach [33].

Coding was performed after the observations were completed, based on the meeting notes taken from February 27th, 2014 to July 31st, 2014. Since it can take time for the people observed to be used to the presence of the researcher [34], and for the researchers themselves to fully understand the domain knowledge of the project [35], the data from the first two meetings were not kept for this study.

Meetings taking place after July 31st were also removed from this analysis. These last meetings were mostly related to deployment activities and featured very little development interactions. While the analysis of the deployment activities

would be interesting, it was decided to keep the development discipline and deployment discipline separate, as the MTS requirements of both disciplines are quite different.

Coding schemes were developed following the Grounded theory approach [33]. In summary, coding was performed using the following steps:

1. Open coding of all entries, going over the data as long as new codes can be added.
2. When no new codes can be added, similar codes are grouped together.
3. Code groups are formalized into schemes.
4. Return to point (1) until no new codes are added and no new schemes can be formed.

After multiple coding iterations, three coding scheme emerged. The first scheme pertains to whether the interaction observation is related to a technical or administrative topic:

Technical: interactions related to technical issues (requirements, bugs, data, etc.),

Administrative: interactions related to administrative issues (deadlines, resources, etc.).

The second scheme pertains to one of the four types of interaction identified:

Team demands (inputs): These interactions are requests made by team members to someone outside the development team.

Team commitments (outputs): These interactions are requests made by someone outside the development team to the team or a team member.

Team coordination (in-out): These interactions are related to meetings which had or will take place between two or more teams on a given issue.

Team liaison (brokering): These interactions are information request to the development team by someone outside the team. The development team cannot answer themselves and therefore act as knowledge brokers with another team.

The third scheme pertains to the type of team interacted with:

Client teams: These teams are responsible for providing requirements and details on what they need the software to do, along with validation of the final result.

3rd party teams: These teams represent the 3rd party library support teams, which performs corrections on the software based on the service-level agreement (SLA) their 3rd party holds with the organization. Two internal module support teams are also included here, as the interaction with these teams followed a protocol similar to the interaction with support teams outside the organization.

Quality teams: These teams are responsible for quality assurance and quality control within the organization.

Ancillary teams within the organization: The organization has many departments, each with their own expertise and technical competencies. For example, one ancillary team was in charge of the creation and configuration of the development and test environments.

In-house development teams: These teams represent other development teams working in parallel projects on the same codebase.

4. Analysis

Data collection returned a total of 464 topics discussed within the 21 weekly status meetings analysed. From these 464 topics, 294 were related to external teams. Therefore, about 60% of all topics discussed were related to requests to external teams, commitments to fulfil for stakeholders, and other interactions that involved external team members.

Figure 1 presents the number of interactions between the observed development team and all external teams. The teams are split based on the five team types presented in the previous section. The closer a team is to the dark centre of Figure 1, the more interactions they had with the observed team, and the closer they were to them. Note that since it was an internal development project for an organization which does not sell software, the actual clients of the package upgraded was the Operations team. The Operations team is in charge of creating and dispatching work orders. Field workers receive the work orders and must on occasion interact with the software. A total

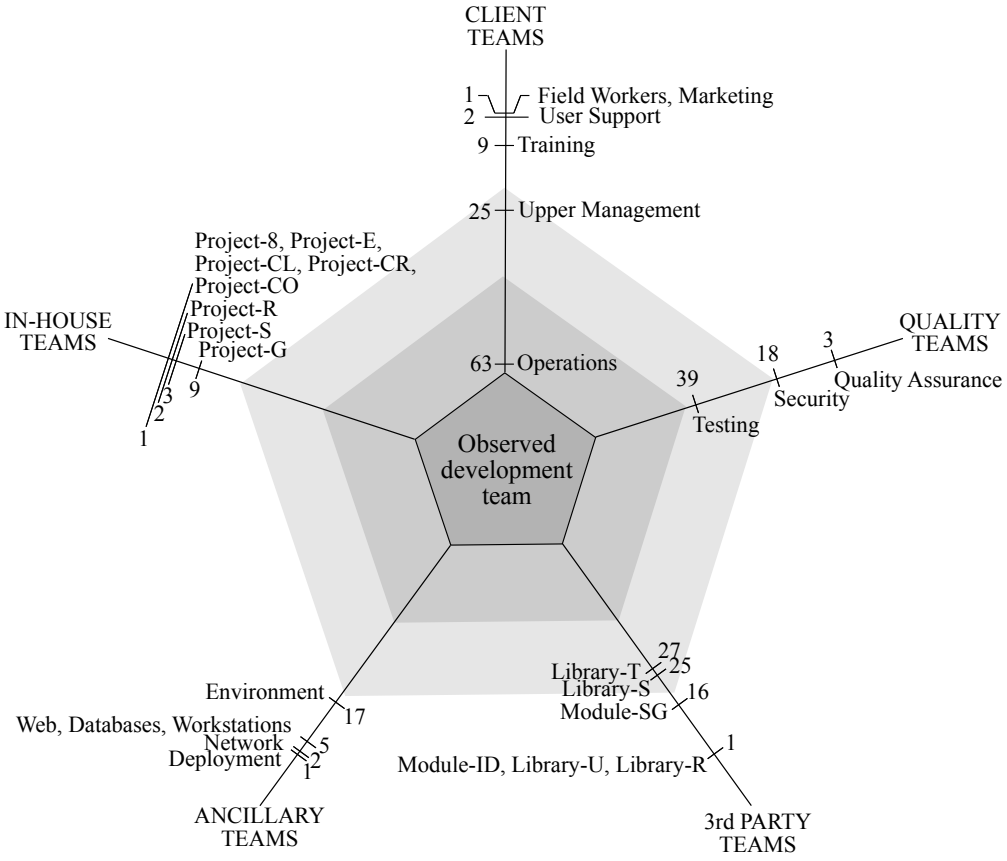

Figure 1. Proximity of each external team with the observed development team. The number of interactions are posted on the axes

of 29 different external teams were contacted during the course of the study.

Table 2 presents the results of the number of interactions with external team members according to their activities, which outlines the amount of interactions and the rationale for interaction (to answer team needs, to fulfil team obligations, etc.). While table 2 present the number of liaison interactions, more details are presented in Figure 2. Table 2 shows that there are numerous administrative as well as technical interactions with all the team categories. Note that eight interactions could not be assigned to a specific team, bringing the total in Table 2 to 294 interactions.

Figure 2 shows the occurrences of liaison interactions between two teams in which the observed development team was involved. It shows that the observed team is pivotal between the client and the quality group. These interactions include requirement clarifications, but also demands by testers to ensure that the

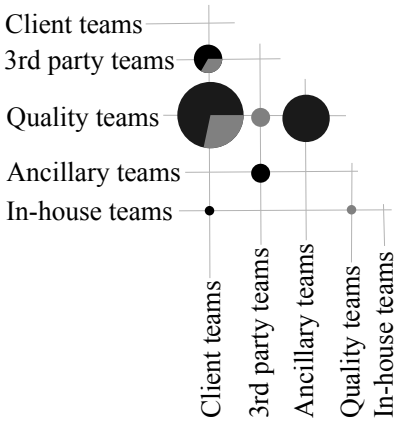

Figure 2. Liaison interactions (knowledge brokering) between external team categories. Bubble size represents the amount of liaison interactions (from one to seven). Black colour represents technical interactions, while grey colour represents administrative interactions

initial data in the system are validated by the clients before testing can start. More details

Table 2. Number of interactions with external teams per team category

Team Category	Team Demands		Team Commitments		Team Coordination		Team Liaison		Total
	Tech	Admin	Tech	Admin	Tech	Admin	Tech	Admin	
Client teams	22	21	19	11	8	7	8	5	101
3rd party teams	35	7	8	3	6	4	4	4	71
Quality teams	3	6	19	7	6	4	10	5	60
Ancillary teams	14	8	3	1	2	0	7	0	35
In-house teams	4	1	8	2	1	1	1	1	19

about the importance of the client/quality interactions can be found in the next section.

4.1. Interaction purpose examples

Tables 3, 4, 5 and 6 present a glimpse of the reasons through actual quotes from the development team. Each table covers one of the four types of interactions. The objective is to give an idea how a topic was associated with the appropriate interaction type and the appropriate external team.

4.2. Failure of the first iteration of the project

As stated earlier, the project observed had already been done once, but failed. A private communication with a manager who witnessed the failure of the first iteration but did not participate in the second one provides some details on the failure. According to the manager, the following factors may have caused the failure of the first iteration:

- Personality conflicts between the development team, the client teams, and the other 3rd party teams. This can be related to "Organizational Skirmish" identified by Tamburri et al. [36].
- Contractual issues between the organization and 3rd party developers. Contract negotiations dragged so long that the contracts were signed moments before the code was scheduled for production.
- Pressure from the project manager to filter interactions with the development team. This manager required that all requests had to be submitted directly to her, resulting in missed or misinterpreted messages. This can be related to the "Radio-Silence" identified by Tamburri et al. [36].
- Documentation mostly incomprehensible by anyone outside the development team. Only the client teams' documentation could be reused as is.

While this statement is only supported by one witness, it still provides some insight as to why the project initially failed.

5. Discussion

This section discusses the results and poses three hypotheses to resolve the identified issues, along with their potential impact on quality.

5.1. First hypothesis: Identification of the critical teams and client implication

This study shows that although interactions with external teams are important, some teams are more important than others. The frequency analysis shows that the interactions of the team loosely follow a Pareto distribution. Approximately 78% of external interactions (229 of 294 interactions) are made from about 28% of all teams contacted (8 of 29 teams). Based on the data in Figure 1, the distribution of these eight teams (categories of the corresponding team in brackets) are:

1. Operations [client team]: 63 interactions.
2. Testing [quality team]: 39 interactions.
3. Library-T [3rd party team]: 27 interactions.
4. Library-S [3rd party team]: 25 interactions.
5. Upper Management [client team]: 25 interactions.
6. Security [quality team]: 18 interactions.

Table 3. Example quotes related to team demands

Demands to	Quote
Client	*"There is a problem with [the client]. We need the configuration data and we have no answer from [the client]. I did some work on this, but I cannot finish by myself."* The team had to ask the client again for the configuration data.
Ancillary	*"Everything has been settled, except for the database configuration. We do not have the access rights [to the environment] to prepare this. [...] This configuration should be done by default! It's like buying a car and not having a key!"* The team had to ask the environment setup team for the rights to change the database configuration.
In-House	*"We just receive an analysis from Project-G, which is about 60 pages. The analysis is very badly written and is essentially incomprehensible."* The team had to ask the Project-G team a clearer document in order to fulfil the analysis.

Table 4. Example quotes related to team commitments

Commitments to	Quote
Client	Upper Management has approved a new project with a high priority and a very aggressive calendar. It is likely that some developers from the development team will be assigned to this new project. The observed development team must finish their current project as soon as possible, as delays will be unacceptable for upper management.
Quality	*"What do we do if we find bugs?"* Quality teams need development support during the developers' holiday, in August. The development team cannot go on holiday all at once: someone must stay in place to correct the bugs found by quality teams.
In-House	The development team must replace a function so it can support true/false/maybe values. This is in order to support Project-R, developed by another team, which will be deployed shortly after their current project ends.

Table 5. Example quotes related to team coordination

Coordination with	Quote
Client	The development team needs the business processes from the client so they can code the appropriate functionalities. But the client expects that the development team will explain how the software will work, and therefore adjust their business process in consequence. There is confusion as to whom is responsible for providing the business processes.
3rd Party	The development team must discuss with Library-T support to determine which changes will be covered under the current contract and which changes will be charged extra to the project.
Quality	The development team pressures the testing team to start acceptance testing even though integrated testing is not finished. The testing team disagrees: the two teams will need to meet afterward in order to decide what to do. *"How can I start acceptance testing if integrated testing only reach 50% success?"*

7. Environment [ancillary team]: 17 interactions.
8. Module-SG [3rd party team]: 16 interactions.

While the other 21 teams have less than ten interactions each.

Therefore, project managers should try to identify the teams most likely to have an impact on the project beforehand, and ensure that communication channels with these teams are clear. In the case observed, this issue was somewhat alleviated by making the testing team sits in the same room as the developers towards the end of the project. They could not do the same with their 3rd party developers, which resulted in some serious issues. For example, communication problems with 3rd party support teams, coupled

Table 6. Example quotes related to team liaison

Liaison between	Quote
From client to quality	The clients need to provide a description of their workflow for the testing team. The testing team are planning acceptance testing and want to design tests which reflect what the client does in its day-to-day work.
From quality to client	A client was assigned to the testing team in order to assist them in their work. However, the client assigned does not answer the telephone or email. The quality team needs to talk to him.
From quality to ancillary	The security team need access to the test environment in order to perform their tests. The Network team needs to open a port for the security team.
From in-house to quality	Testers need to know if they need to perform testing for the integration of Project-G within the current project. So far, the in-house team developing Project-G has not answered.

with poor service-level agreements (SLA), required multiple reworks of some simple change requests, each taking one month to perform [22].

The Pareto analysis shows that the clients, the Operations team, is by far the external team most contacted. However, the development project followed a waterfall approach, with fixed requirements. Why so many interactions are needed with the clients if the requirements are fixed since the beginning? Many details and subtleties became evident as the developers progressed into the project. Some requirements have emerged or have changed very late during the project. Some of these changes were client requests, but others were tasks that the client needed to do.

For example, since this project is related to the update of an old package, some of the new databases must be updated with the data already in the old package. However, a lot of the data in the old package are obsolete: dropdown menu items are no longer used; database columns are no longer filled, etc. The developers cannot know these subtleties, and rely on clients to tell them which data to port to the new package, and which data to remove. In this case, the clients did not have the resources to do this task for the developers, which leads to multiple delays.

This shows that all projects, whether Agile or disciplined, require continuous interactions with stakeholders. But while Agile principles emphasize flexibility to clients' needs ("our highest priority is to satisfy the customer" [37]), this study shows that the clients must also be flexible to developers. Clients have obligations to fulfill. The domain knowledge of the clients was very important in this project. Some delays can be attributed to the unavailability of the client or to late responses to critical requests. The clients were required to provide many details about what the old package did, and why the old package worked that way, and on what the client wants for the new package. The clients' technical expertise was limited, but they knew very well their workflow and how they want the future application to merge with this workflow.

For project managers, we propose that any client/provider agreement ensures that the client is willing to actively help developers. For example, the simple task of seeding a database with its initial dataset is difficult to plan ahead: it can be done once the database structure is completed, using data that is usually provided by the client. In this study, delays in obtaining clients responses have led to delays in database configuration, which caused the tests to start late, and ultimately to be shorter than planned.

Previous Agile studies have used client delegates to ensure coordination between the real client and the development teams [11, 14]. Delegation of client duties can prevent constant interruption of the workflow of developers. One study assigned each team with "a support person".

> Supporting the customer using all the solutions that the team had to provide was a critical task. This required a vast amount of knowledge of all moving pieces. Before we had the support person, the customers interrupted subject matter experts [i.e. developers] directly. The subject matter experts typically

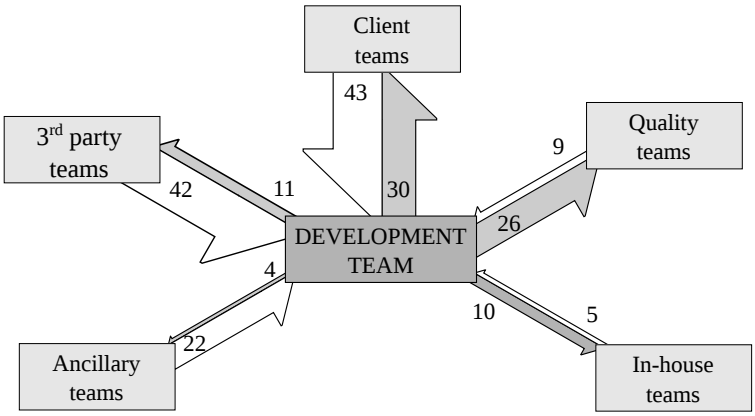

Figure 3. Chain of commitments between teams. White arrows indicate answers to development team demands. Grey arrows indicate team commitments that the development team must fulfil

dealt with too many support requests and ended up context switching in and out of the tasks at hand [11].

The impact on quality in the case observed is mainly transcribed in terms of delays. The failure to provide answers in a due manner to the questions of the development team led to multiple delays. In this case, these delays cause the testing phase to be greatly reduced. In addition, some code written by 3rd parties could not be reviewed in time for delivery and was included in the codebase as-is. By the accounts of the developers themselves, a lengthy support process will be necessary post-delivery to ensure that all the issues are sufficiently smoothed out.

5.2. Second hypothesis: Developers as knowledge brokers within the MTS

Figure 3 illustrates the two-way interactions between external teams and the development team, based on the team demands and team commitments found in Table 2. For example, in the interactions between the development team and the client teams, the development team had 30 commitments (grey arrow) toward the client teams, while the client teams answered 43 demands (white arrow) from the development team.

The left hand side of Figure 3 shows the team categories with a majority of demands from the development team (large white arrows), while the right hand side shows the team categories with a majority of commitments from the development team (large grey arrows). The client teams, being fairly balanced in demands and commitments, remains in the middle. What should be seen from Figure 3 is that demands flows from the left to the right. Ancillary teams fulfil developers' requests, so that developers can fulfil quality teams' requests.

Here is an example taken from the interactions observed. The quality teams needed many test environments in order to perform their work (acceptance environment, load testing environment, etc.). The development team was therefore committed into building these environments and ensuring that they were coherent with the latest available versions of the package and that they were stable enough to support test activities. While they could do some of the work themselves, they needed the support of the environment setup team, an ancillary team. However, the environment setup team did not fulfil its commitment appropriately, causing a number of issues to the development team. These environmental issues cause the development team to fail in some of their commitments toward the quality teams, causing delays and ultimately, the cancellation of some of the test activities.

Some of these relationships might seem self-evident, but others might not be as well-known. As presented in our previous paper [22], managers should be wary of other projects imposing changes to the current project. Project managers should also ensure that all relevant teams (3rd party teams, ancillary teams) are ready to help the

development team. In the case presented above, many issues stemmed from poor communications between the development team and the environment team.

The role of the development team in this case is that of a broker. Developers need to redirect the requests they receive to the appropriate team. To take an analogy from the TCP/IP protocol, the development team is the default gateway for the external teams. External teams needing something related to the project will ask the developers first, which will then redirect the team to the appropriate resource when necessary.

This is especially true of the relationship between clients and testers. Clients and testers do not know how the application was built, who was contacted to code the software, what are the dependencies. They are mostly conscious on what they see on their end. When something goes wrong, their only contact is the development team. Clients and testers need some answers but do not know who to ask; developers know and must assist them.

For project managers, this study shows that testers cannot work efficiently if they are kept completely isolated from the development team. Testers need to ask many questions in order to perform their work, and these questions must be efficiently relayed to the appropriate external team. In the case studied, toward the end of the project, management had the testing team sits directly with the development team. Their goal was to diminish bug resolution times, but it also helped the testers in the setup of the different testing phases and testing environment (integration, acceptance, load, and deployment). The same can be applied to clients. While it might not make sense to put the client in contact with every relevant external team, clients' questions can be distributed by the developers to the relevant external teams.

The need for knowledge brokers have been identified in the literature [11, 25, 36]. It is sometimes identified as a "coordinator role" [16].

> Brokers are those individuals who link disconnected subgroups. [Another study] found that system-level coordination is achieved more efficiently when certain key individuals connect different subgroups as opposed to when all individuals are directly connected to one another. Complex MTSs may be more efficiently coordinated if certain individuals act as ambassadors by connecting their team to others within the system [25].

The question of whom to assign to the role of knowledge broker varies from study to study however. It should be someone who has a widespread knowledge of the system [10, 11]. It is, however, unnecessary to have a broker between each team. As presented in the previous section, teams with a potential critical impact on the development team's work should be identified. Knowledge brokers can therefore be assigned only for those critical teams [38]. Minor teams and modules could be more isolated from the development team under study.

The impact on quality rests on the fact that the development team does not work in isolation. There are many other teams working indirectly on the project which require adequate support to perform their work. Here are a few examples:

- Testers need to obtain real data from the clients in order to perform tests that can be relatable to what goes on in reality.
- Testers need working testing environment with an up-to-date code in order to perform adequate tests.
- Third party support teams need to know the type of tests to be performed in order to ensure that their infrastructure will support these tests (e.g. stress testing or security testing cloud storage services).

Failure to relay the needs of one external team to another can lead to the cancellation of important activities.

5.3. Third hypothesis: Managing technical and administrative interactions

Before this study, the organization managers and the development team were convinced that their meetings were mostly administrative; discussing deadlines, budget and resources. Observations proved that most of these discussions were actually technical and involved bugs, issues, design,

solutions, etc. It is therefore not surprising that most of the interactions with external teams are also technical in nature.

But this information should not be exchanged only from one manager to another. This study's suggestion to project managers is to make sure that developers in different teams are able to talk to each other. Managers have a tendency to protect their developers from outside interference, and it is good to keep an eye on that, as this was an issue with Upper Management in this case [22]. But developers also need to be able to obtain technical information from other teams, and to plan technical solutions and strategies together.

The literature recommends a layered structure where the lower levels are able to share technical details, while the higher levels are able to share the administrative big picture [10, 11].

> Cross team knowledge sharing is difficult. [...] After 1.5 year into practicing Agile, we found the best way to mitigate, is to have weekly Scrum of Scrums (S2) meetings and daily tech leads stand-up meeting. For the stakeholders, Scrum of Scrums of Scrums (S3) was very helpful to get things prioritized [11].

The impact on quality is that technical issues facing the whole codebase are not discussed anywhere. Individual teams might be aware of the issues, but without a platform to discuss and voice their concern, these issues remain latent and unaddressed. Organizations have administrative strategies, where managers discuss future plans and projects, but how many of them have technical strategies, where engineers can discuss future maintenance challenges and issues?

5.4. Threats to validity

A threat to the validity with the use of a single study is the generalizability of its conclusions. The objective of this study was however not to build a theory applicable to all software development projects, but to identify new potentially interesting practices and issues from the industry. While this study is limited to a single case, it nonetheless presents new qualitative and quantitative data showing the role of clients during development, the role of developers as knowledge brokers, and the importance of technical coordination at the MTS level.

Proper case study practices recommend triangulating the data, that is to obtain data from different approaches in order to confirm the conclusions [39, p. 97]. For instance, conclusions made through observations can be confirmed with interviews and artefact analyses. In this case, it was not possible to access any other data source, limiting the work to an exploratory study instead of a fully fledged case study. That is why the recommendations are presented as hypotheses to be tested, instead of solutions.

6. Conclusions and further works

This exploratory study shows the impact interactions within the multi team system can have on project success. Due to the single study nature of this research, future research should look into whether the three hypotheses presented herein are relevant in other cases.

1. Identify the external teams most likely to have an impact on the development project, based on a Pareto analysis and ensure proper communication channels with the most important ones. Otherwise, slow communications will cause delays during development, which might result in rush development work and shorter testing time.
2. Ensure that knowledge brokers exist within the development team to redirect requests from one external team to the proper other external team. Otherwise, some activities with an indirect impact on the development project (e.g. testing) might be in jeopardy.
3. Ensure that discussion platforms at the multi-team level are not limited to administrative issues. Technical solutions and strategies must be discussed between teams. Otherwise quality issues affecting the whole codebase could remain unaddressed.

Project managers should be aware of the impact of multi team systems on their projects.

From a disciplined, plan-driven approach, to an Agile, people-driven approach, there is a need for an integrated, organization-driven approach, where the team is integrated within its organization. Teamwork experts have recommended breaking the isolation between individuals in order to ensure that the whole team works together. We should now see if these recommendations hold at the organization level, in order to ensure that the whole organization works together. There might be no "I" in "team". But how much place for "us" and "them" are we willing to work with within the organization?

7. Acknowlegments

This research would not have been possible without the agreement of the company in which it was conducted, which prefers to stay anonymous, and without the generous participation and patience of the software development team members from whom the data were collected. To all these people, we extend our grateful thanks.

This work was supported by the Natural Sciences and Engineering Research Council of Canada, under grant number A-0141.

References

[1] J. Porck, "No team is an island: An integrative view of strategic consensus between groups," Ph.D. dissertation, Erasmus University Rotterdam, 2013.

[2] F.Q. da Silva, A.C.C. França, M. Suassuna, L.M. de Sousa Mariz, I. Rossiley, R.C. de Miranda, T.B. Gouveia, C.V. Monteiro, E. Lucena, E.S. Cardozo, and E. Espindola, "Team building criteria in software projects: A mix-method replicated study," *Information and Software Technology*, Vol. 55, No. 7, 2013, pp. 1316–1340.

[3] R.A. Guzzo and M.W. Dickson, "Teams in organizations: Recent research on performance and effectiveness," *Annual Review of Psychology*, Vol. 47, 1996, pp. 307–338.

[4] R.A. Guzzo and E. Salas, *Team Effectiveness and Decision Making in Organizations*. Wiley, 1995.

[5] W.S. Humphrey, "The Team Software Process (TSP)," Software Engineering Institute, Pittsburgh, PA, USA, Tech. Rep. ESC-TR-2000-023, 2000. [Online]. https://www.sei.cmu.edu/reports/00tr023.pdf

[6] V. Santos, A. Goldman, and C.R.B. de Souza, "Fostering effective inter-team knowledge sharing in Agile software development," *Empirical Software Engineering*, Vol. 20, No. 4, 2015, pp. 1006–1051.

[7] M. Paasivaara, C. Lassenius, and V.T. Heikkilä, "Inter-team coordination in large-scale globally distributed Scrum: Do Scrum-of-Scrums really work?" in *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '12. New York, NY, USA: ACM, 2012, pp. 235–238.

[8] A. Martini, L. Pareto, and J. Bosch, *Improving Businesses Success by Managing Interactions among Agile Teams in Large Organizations*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 60–72.

[9] C.M. Tartaglia and P. Ramnath, "Using open spaces to resolve cross team issue [software development]," in *Agile Development Conference (ADC'05)*, 2005, pp. 173–179.

[10] H. Smits and G. Pshigoda, "Implementing scrum in a distributed software development organization," in *Agile 2007*, 2007, pp. 371–375.

[11] E.C. Lee, "Forming to performing: Transitioning large-scale project into Agile," in *Agile 2008 Conference*, 2008, pp. 106–111.

[12] J. Sutherland, G. Schoonheim, and M. Rijk, "Fully distributed Scrum: Replicating local productivity and quality with offshore teams," in *2009 42nd Hawaii International Conference on System Sciences*, 2009, pp. 1–8.

[13] R.P. Maranzato, M. Neubert, and P. Herculano, "Moving back to Scrum and scaling to Scrum of Scrums in less than one year," in *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*. New York, NY, USA: ACM, 2011, pp. 125–130.

[14] M. Paasivaara and C. Lassenius, "Scaling Scrum in a large distributed project," in *2011 International Symposium on Empirical Software Engineering and Measurement*, 2011, pp. 363–367.

[15] P.L. Bannerman, E. Hossain, and R. Jeffery, "Scrum practice mitigation of global software development coordination challenges: A distinctive advantage?" in *2012 45th Hawaii International Conference on System Sciences*, 2012, pp. 5309–5318.

[16] D.E. Strode, S.L. Huff, B. Hope, and S. Link, "Coordination in co-located Agile software de-

velopment projects," *Journal of Systems and Software*, Vol. 85, No. 6, 2012, pp. 1222–1238, special Issue: Agile Development.

[17] A. Mundra, S. Misra, and C.A. Dhawale, "Practical Scrum-Scrum team: Way to produce successful and quality software," in *2013 13th International Conference on Computational Science and Its Applications*, 2013, pp. 119–123.

[18] A. Scheerer, T. Hildenbrand, and T. Kude, "Coordination in large-scale Agile software development: A multiteam systems perspective," in *2014 47th Hawaii International Conference on System Sciences*, 2014, pp. 4780–4788.

[19] A. Scheerer, S. Bick, T. Hildenbrand, and A. Heinzl, "The effects of team backlog dependencies on Agile multiteam systems: A graph theoretical approach," in *2015 48th Hawaii International Conference on System Sciences*, 2015, pp. 5124–5132.

[20] A. Martini, L. Pareto, and J. Bosch, "A multiple case study on the inter-group interaction speed in large, embedded software companies employing Agile," *Journal of Software: Evolution and Process*, Vol. 28, No. 1, 2016, pp. 4–26, jSME-14-0083.R3.

[21] D.A. Tamburri, R. Kazman, and H. Fahimi, "The architect's role in community shepherding," *IEEE Software*, Vol. 33, No. 6, 2016, pp. 70–79.

[22] M. Lavallée and P.N. Robillard, "Why good developers write bad code: An observational case study of the impacts of organizational factors on software quality," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1, 2015, pp. 677–687.

[23] M.A. Marks, L.A. DeChurch, J.E. Mathieu, F.J. Panzer, and A. Alonso, "Teamwork in multiteam systems," *Journal of Applied Psychology*, Vol. 90, No. 5, 2005, pp. 964–971.

[24] J.E. Mathieu, M.A. Marks, and S.J. Zaccaro, *Multi-team systems*. London: Sage, 2001, pp. 289–313.

[25] R. Asencio, D.R. Carter, L.A. DeChurch, S.J. Zaccaro, and S.M. Fiore, "Charting a course for collaboration: A multiteam perspective," *Translational Behavioral Medicine*, Vol. 2, No. 4, 2012, pp. 487–494.

[26] M. Cohn, Advice on conducting the Scrum of Scrums meeting, (2007). [Online]. https://www.scrumalliance.org/community/articles/2007/may/advice-on-conducting-the-scrum-of-scrums-meeting Retrieved 2015-08-21.

[27] Z.U.R. Kiani, D. Smite, and A. Riaz, "Measuring awareness in cross-team collaborations – Distance matters," in *2013 IEEE 8th International Conference on Global Software Engineering*, 2013, pp. 71–79.

[28] T. Chau and F. Maurer, *Knowledge Sharing in Agile Software Teams*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 173–183.

[29] K. Lanaj, J.R. Hollenbeck, D.R. Ilgen, C.M. Barnes, and S.J. Harmon, "The double-edged sword of decentralized planning in multiteam systems," *Academy of Management Journal*, Vol. 56, No. 3, 2013, pp. 735–757.

[30] J.M. Verner, O.P. Brereton, B.A. Kitchenham, M. Turner, and M. Niazi, "Systematic literature reviews in global software development: A tertiary study," in *16th International Conference on Evaluation Assessment in Software Engineering (EASE 2012)*, 2012, pp. 2–11.

[31] M. Lavallée and P.N. Robillard, in *2015 IEEE/ACM 3rd International Workshop on Conducting Empirical Studies in Industry*, 2015, pp. 12–18.

[32] J.P. Looney and M.E. Nissen, "Organizational metacognition: The importance of knowing the knowledge network," in *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, 2007, p. 190c.

[33] A.L. Strauss, *Qualitative Analysis for Social Scientists*. Cambridge University Press, 2003.

[34] H.A. Landsberger, *Hawthorne Revisited*. Cornell University, 1958.

[35] T.C. Lethbridge, S.E. Sim, and J. Singer, "Studying software engineers: Data collection techniques for software field studies," *Empirical Software Engineering*, Vol. 10, No. 3, 2005, pp. 311–341.

[36] D.A. Tamburri, P. Kruchten, P. Lago, and H. van Vliet, "Social debt in software engineering: Insights from industry," *Journal of Internet Services and Applications*, Vol. 6, No. 1, 2015, p. 10. [Online]. https://jisajournal.springeropen.com/articles/10.1186/s13174-015-0024-6

[37] P. Runeson, A. Stefik, and A. Andrews, "Variation factors in the design and analysis of replicated controlled experiments," *Empirical Software Engineering*, Vol. 19, No. 6, 2014, pp. 1781–1808.

[38] R.B. Davison, J.R. Hollenbeck, C.M. Barnes, D.J. Sleesman, and D.R. Ilgen, "Coordinated action in multiteam systems," *Journal of Applied Psychology*, Vol. 97, No. 4, 2012, pp. 808–824.

[39] R.K. Yin, *Case Study Research: Design and Methods*, ser. Applied Social Research Methods, L. Bichman and D.J. Rog, Eds. Thousand Oaks, CA, USA: Sage, 2002, Vol. 5.