2011 Sixth IEEE International Conference on Global Software Engineering

The impact of Multi-site Software Governance on Knowledge Management

Christina Manteli
VU University Amsterdam
cmanteli@cs.vu.nl

Bart van den Hooff
VU University Amsterdam
bhooff@feweb.vu.nl

Antony Tang
VU University Amsterdam
atang@cs.vu.nl

Hans van Vliet
VU University Amsterdam
hans@cs.vu.nl

***Abstract*—Software Development Governance (SDG) is an emerging field of research, under the umbrella of information technology governance. SDG challenges increase when software development activities are distributed across multiple locations. Coordination of knowledge management processes requires specific attention in multi site development. This paper outlines a multi-site software governance structure, based on three aspects: the business strategy that binds the relationship of the remote offices, the structure and composition of the remote teams and the way tasks are allocated across sites. Knowledge management processes (including knowledge creation, knowledge transfer and communication) are identified and the influence of different governance structures on these processes is discussed. We do so through a case study at Océ, a multi-national company in printing systems.**

***Keywords*-Knowledge management; multi-site software development; software governance;**

I. Introduction

Governance is defined as those arrangements and practices that an organization puts in place to ensure that the activities are appropriately managed [1]. In fact, only recently attention has been paid to define governance in software development: what are the structural attributes of software development governance and what are the coordination mechanisms that this governance embraces. SDG challenges increase when software development activities are distributed across multiple locations [2]. For example, Heeks et. al. [3] investigate different strategies in multi-site cooperation that can lead to either *synching*, that is a successful cooperation, or *sinking*, an unsuccessful cooperation.

With the continuous and evolving strategies of distributed software development, scholars have been considering the challenges and the key issues of managing the knowledge in those distributed environments (e.g. [4], [5], [6]). For example, it was found that communication frequency and speed significantly drops among remote teams [7]. As a result, knowledge tends to "stick" to locations and it becomes difficult to transfer that knowledge from one site to another [8].

In this paper, we argue that the way software companies govern their multi-site development activities, can impact the management of knowledge across remote locations. We investigate two different types of multi-site software governance through a case study, in a multi-national organization. We identify several knowledge management challenges that emerge in these distributed environments, and we observe how the governance structures impact these knowledge management challenges.

In section II, we define the attributes of multi-site software governance, and we elaborate on the background theory of knowledge management issues in distributed environments. In sections III and IV the project overview and the research methodology are described. In section V, we present the knowledge management challenges identified within the case study, and in section VI the impact of the multi-site software governance structures on knowledge management is discussed. Finally, section VII summarizes the main lessons learned from this research and we conclude with suggestions for future research.

II. Background Theory

A. Multi-site Software Development Governance

Software Development Governance (SDG) is an emerging field of research, under the umbrella of information technology governance. The goal of SDG is to ensure that business processes of the software company meet the strategic requirements of the organization [9]. In order to achieve this, software development processes should be aligned with the company's business goals on a strategic, tactical and operational level. Chulani et. al. [9], for instance, describe a unified view of SDG from the start at the senior executive level and all the way down to the practitioner level where projects are implemented.

The challenges of SDG increase, when software development activities are engaged in a multi-site environment. Several proposals have been made to define the attributes of a software governance model for distributed development projects, and the coordination mechanisms that this model should embrace. Ramasubbu and Balan [2] have proposed a research model on how to create a governance framework for distributed software development, focusing on three stages of a project lifecycle; planning, execution and reflection. Gewald and Helbig [10] suggest a governance framework for managing outsourcing engagements based on organizational structures, joint processes and relationship management functions.

As Bannerman [1] suggests, SDG should include both a functional perspective in terms of *what governance does* as

978-0-7695-4503-5/11 $26.00 © 2011 IEEE
DOI 10.1109/ICGSE.2011.16

IEEE computer society

well as a structural perspective dealing with *what governance looks like*. In this paper, we make our contribution and define a multi-site software development governance model from a structural perspective. We outline a multi-site software governance structure, based on three aspects: the business strategy that binds the mutual relationship of the remote offices, the structure and composition of the remote teams and the way tasks are allocated across sites. In the rest of this section, we elaborate more on each of those attributes and the reasons why we define them in the context of multi-site software governance.

1) Business Strategy: According to [10] a multi-site governance model should steer and control the cooperation of the remote offices, based on partnership and mutual trust. Additionally, Carmel and Tija [11] note that one of the things that companies should not forget when they operate in a global environment is the broader strategic goals and their legal implications. Multi-site business strategy should therefore consider the contractual and legal relationships between the remote sites and the implications these might have in their collaboration, communication and knowledge management process.

2) Team Structure and Composition: In software development, team structure and composition is a critical factor for good performance [12]. Team size, role descriptions and role distribution are among those characteristics in distributed teams that can influence team coordination and communication and therefore team performance (e.g. [7], [13], [14], [15]). For example, Kofman and Klinger [16] suggest that role confusions (meaning the confusion caused by the difference between *how the role is described* and *what people actually do within the role*) may affect team performance and communication. Faraj and Sproull [17] also propose expertise coordination of a team – that is the coordination of knowledge and skill dependencies – as an important component of teamwork coordination for knowledge teams. In this paper we argue that team structure and composition is an integral part of multi-site software development and it should be considered part of the governance structure.

3) Task Allocation: Several criteria exist on how to distribute work across sites, such as based on the area of expertise, on the software architecture and structure, or even based on the development process steps from requirements elicitation and communication to maintenance [18]. Work dispersion and the interdependencies between the distributed tasks influence the coordination and administration of the remote offices [7]. For example, Mockus and Weiss [19] argue that when assigning a task to a remote site, it is important to consider that the allocated task matches the development-resource capacities of that location. Additionally, according to Carlile [20], as the number of dependencies between actors increases, the complexity and the amount of effort required to share and access knowledge at a boundary also increases. Hence, we consider task allocation as a fundamental attribute of multi-site software governance.

B. Knowledge Management Challenges

Software development is a knowledge-intensive process and knowledge management challenges increase when the development activities are distributed across multiple locations. Managing the knowledge in multi-site software development includes the coordination of communication between the remote teams [21]. Distance, for instance, between the teams introduces barriers to informal and face-to-face communication, and the collaboration of the remote colleagues is dependent on synchronous or asynchronous communication tools [22]. Additionally, communication speed and frequency is influenced by the coupling and the interdependencies among the distributed tasks. It is suggested for example, that tightly-coupled tasks that are distributed across multiple sites, can increase the communication frequency [19].

Another knowledge management challenge in distributed software development is the knowledge capture which is seen as the process of recording knowledge in a medium, that is transforming and encoding it as information [23]. According to Correia and Aguiar [23], the effectiveness of how knowledge is captured into artifacts, and acquired by other team members, is of crucial importance to a project's success. Research also suggests that in distributed software development, documentation must be current and reflect what various teams are using and working on, to prevent assumptions, misunderstandings and to support maintainability [24].

Moreover, several factors can impede knowledge transferability across sites such as the use of different working methods or the differences in skills and expertise of the remote colleagues [25]. For example, Carlile [20] argues that as the differences in the amount and the type of knowledge that people possess increases, the effort required to share and transfer that knowledge also increases. Furthermore, transactive memory – the knowledge of *who knows what* and *where knowledge resides* [26] – has been studied as a supporting structure in knowledge transfer across multiple locations [25]. Kotlarsky and Oshri [27], for instance, investigated two globally distributed system development projects and argue that transactive memory as a mean of knowledge sharing contribute to successful collaboration across remote teams.

In this paper, we investigate knowledge management challenges that can emerge in a multi-site software development environment. Based on the aforementioned literature we classify the identified challenges under three main categories, communication, knowledge creation and storage and knowledge transfer. We also make a distinction between system-generic and unit-specific knowledge. We define system-generic knowledge as the comprehensive

knowledge of the entire system that teams are working on. In other words, it is the knowledge of how the end product looks and functions. Unit-specific knowledge is the particular knowledge that the individual has, for the specific unit he or she is working on.

III. Project Overview

The research was conducted at Océ[1], a multi-national company in printing systems, part of the Canon Group. Océ is headquartered in the Netherlands, with offices in more than 100 countries and over 20,000 employees. Research and development departments work on hardware as well as software innovations, and are located in nine different countries.

Océ successfully applies an agile development methodology to encourage creativity and productivity. The organization is flat and employees are encouraged to be proactive in owning up to work responsibilities. Océ has deliberately opted for cooperation, as opposed to hierarchy, to foster innovation and entrepreneurship. People are not curtailed by strict processes and the drive to deliver business results is strong, so the latter will take precedence over writing excessive documentation, and a lot of knowledge therefore remains transactive.

This case study focuses on one business unit which specializes in high-end printers. The development of the software used in those printers is distributed among the main site (site NL) and two remote sites: site A and site B. The software includes units such as accepting requests, controlling print jobs, rendering images, controlling devices (e.g. scanners) as well as local and remote user interfaces.

In the next subsections, we first sketch typical team structure at Océ, followed by the business strategy, team structure, and task allocation in the site NL - site A and site NL - site B coalition, respectively.

A. Project team structure

In a typical team structure of project teams at Océ, a project manager heads the project and is responsible for its planning, realization, and successful completion. The project manager also agrees upon the high-level specifications of the project with upper management and marketing personnel. Requirements and specifications are compiled into product properties by the lead architect in the team. The specifications written by the lead architect start from a user-centric view, i.e., scenarios on how the end-users will interact with the product. For instance, the architect is responsible to decide what happens in case of a request from the user of the printer as in how should it function, which software units should be triggered and how should they function, define the interfaces between these units and the like. Teams also comprise a system integrator who integrates the different software units to build the software system. Additionally, the system integrator reports issues encountered during integration and assigns them to the appropriate team or person to be addressed. All three – the project manager, the lead architect, and the system integrator – are assigned to a project for its entire duration until the product has been released.

Project teams also comprise one or more software unit teams that implement the software units. Each unit has a unit leader and a unit architect analogous to the project manager and lead architect at the project level. The unit leader is responsible for planning and organizing. The unit architects transform the high-level specifications received from the lead architect into detailed technical specifications and pass them to the software engineers who implement the code and test it. The unit architects are coordinated by the lead architect, who is often the only team member with the overall view of where (or in which units) do the different functionalities of the product reside. Most software units are not developed for a single product but their deliverable is tuned and integrated into several products. A software team can develop a software unit for four, or even more, projects at the same time. This challenges system behavior as well as architecture.

B. Relationships between site NL and site A

The collaboration between site A and site NL concerns the co-development of a software unit, which means that the development activities of that unit are distributed among members in a team in the NL site and members in a team in site A. Not all information available at site NL can be freely shared with site A.

For the unit developed between site A and site NL, the unit leader is located in site NL. In site A, another role is created, referred to as the team leader, who is the coordinator of the local team at site A as well as the main contact person with the team in site NL. Both teams consist of software engineers, testers as well as unit architects.

Development tasks are allocated to the software engineers of the two sites by the unit leader of the specific software unit. Two main criteria are applied in deciding which tasks should be sent to site A: the first is based on previous experience, i.e., whether a software engineer has worked on a certain task before, and therefore he or she is more capable in dealing with a specific change request, resolving related defects or adding new features. The second criterion is the complexity of the task. If requirements are difficult to communicate through email, or via the phone, the tasks are more likely to be assigned to someone at the NL site.

C. Relationships between site NL and site B

Site B is responsible for the development of a software unit, with an independent unit team that develops the unit and ships it back to site NL for integration. The team in site B has a unit leader, a unit architect and software engineers

[1] www.oce.com

and testers. All information available at site NL can be shared with site B.

In the beginning of each release cycle, the project manager from the NL site and the unit leader from site B, as well as the lead architect from the NL site and the unit architect from site B, create a plan for the next release and discuss the requirements to be implemented. At the end of each release, the work developed and tested in site B is shipped back to the NL site for integration and system and product testing.

IV. Research Methodology

The research was conducted based on a qualitative data analysis approach. Qualitative research refers mainly to the investigation and analysis of personal experiences and behaviors, as well as organizational functions and social interactions [28]. In order to gather the required data for the analysis, we chose to perform semi-structured interviews. In semi-structured interviews, questions can be open-ended allowing a conversational manner, while at the same time, an interview protocol can still be followed [29].

To gather the data, 20 interviews were conducted with a duration of approximately 90 minutes each. An interview protocol was designed to guide the discussions, which covered questions on the topics of communication and knowledge management. The respondents included employees from all three sites, with different roles and positions. Table I presents the number of the respondents, their roles and the location. Site A has no system integrators, unit architects or designers, therefore no interviews from these roles at that location were held.

Table I
Interview Participants across locations

	site NL	site A	site B
Software Engineers, Testers, Unit Architects	5	3	2
System Integrators, Unit Architects, Designers	3	-	1
Project Managers, Team Leaders	2	2	2
Total	10	5	5

All interviews were recorded and transcribed. To process the data, the Atlas.ti[2] tool was used, a commercial software for qualitative analysis of textual and visual data. The interviews were analyzed based on the coding process of microanalysis or otherwise called a "line-by-line" analysis [28]. The codes were eventually grouped into emerging concepts. These concepts reflect the identified knowledge management challenges and the multi-site governance structures that impact those challenges. In a companion article [36], a larger set of interviews is used to validate a collection of software architecture knowledge management practices.

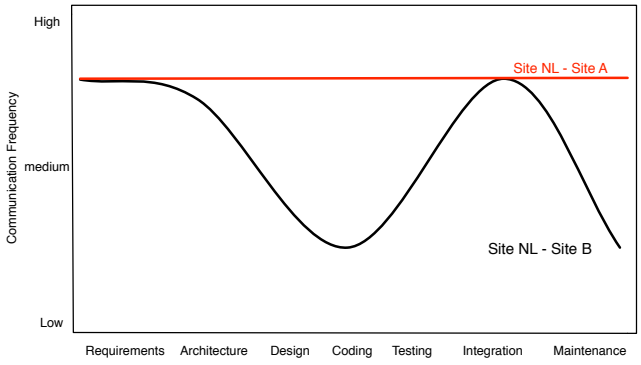
[2]www.atlasti.com

V. Knowledge Management Challenges

A. Communication

Within the case study company, several observations can be made with regard to communication. Between site NL and site A, communication frequency is more intense among all the team members and across all development phases, from software engineers to unit leaders. Between site NL and site B communication frequency depends on the development phase. More specifically, a more intense communication seems to be necessary in the beginning of the release cycle during requirements planning and communication, as well as near the end of a release cycle during integration and testing. Figure 1 qualitatively illustrates the communication frequency of site NL - site A and site NL - site B coalition.

Figure 1. Communication Frequency through the development life cycle



In a distributed development environment, communication can be either asynchronous using emails and chats, or synchronous through video conferences and face-to-face meetings. Synchronous communication is generally considered preferable in distributed environments [30]. The reason is that a small issue can take longer to be resolved using emails circulating between sites, while a brief conversation through the phone can quickly resolve and clarify issues. Communication speed and frequency is important in multi-site environment, and potential delays in communication can cause delivery and project delays.

Based on the need for communication among the different roles and across locations, the use of the different means of communication varies. Software engineers and architects use an instant messaging (IM) tool on an everyday basis. The IM tool allows for direct exchange of messages as well as the possibility for desktop sharing. This synchronous communication part of the chat tool covers the need for the everyday communication that software engineers and architects have. The introduction of the instant messaging (IM) tool was a significant improvement in the communication between site A and site NL, during the last year. Through the IM tool, they can see when some of their remote colleagues are online and available

at the moment, and they come directly in contact with the person they need. Considering the high needs for frequent communication that the two locations have, the addition of a complementary tool of synchronous communication was perceived from the Océ employees as a great advancement in the collaboration methods.

"The advantage of chat is that it is much faster. Once the person is available, you start asking and you can do this very fast in a short time" (Team leader, site A)

With the IM tool, software engineers can also use desktop sharing, improving communication speed and potentially improving performance. Desktop sharing permits remote colleagues to have a real time, interactive communication while they share their desktop environments. They can check each other's work, if certain features have been implemented the right way, or solve bugs by tackling problems together and assisting each other.

"And we can also talk and give control and then you can go sort of around the code and check and it's very fast. Desktop sharing covers pretty much all the needs I have."(Software engineer, site A)

"We share the desktop of a developer - like this morning we did it again to just get a feedback on the look and feel of an implemented part or behavior - and then you can, together with the developers in site A and site NL, you can click through stuff and see what has been implemented before and check in the code." (Unit leader, site NL)

In the case of unit leaders, phone calls and organized video conference are the most common means of synchronous communication. The difference is that unit leaders, although they still need to communicate on an everyday basis, hardly ever work all day in front of their desk. Chatting therefore cannot serve their needs, and phone calls are preferred. Email seems to be the next favorite tool for communication for all roles. It serves best when many people need to be involved in a discussion, or informed about a decision. At the same time, email allows for a more direct means of sharing documents with many colleagues. Finally, traveling is more common for unit leaders, and less so for software engineers.

B. Knowledge Creation and Storage

According to Hansen et. al. [31], codification is pursued when knowledge is documented and stored in databases, while personalization relies on the tacit knowledge that people possess and sharing that knowledge in person-to-person communications. With their choice for an agile development method, the NL site has deliberately opted for a strong personalization strategy. Employees are encouraged to cooperate with each other, taking initiative and responsibility without being constrained by strictly defined processes. With this approach, much of the knowledge remains tacit and details will not always be updated in the corresponding documents.

"*And sometimes there are documents, the system behavior documents, and system architecture documents, but they are mostly on a higher level.*" (Software Engineer, site NL)

"*Most of the time you don't just look for information in documents or in databases, but you walk to a person whom you know is working on stuff or knowledgeable and you just ask questions. And maybe that person also doesn't know but he will give you a pointer to somebody else, and so you just do it by communicating and do it by talking to people.*" (Unit Leader, site NL)

The observed codification and personalization practices within site NL have certain implications for site A and site B. In the case of site B, we noticed that documents are stored and shared with site NL through the use of different repositories and tools. These tools include development platforms and applications as well as communication tools like emails and chats. Consequently, people find it difficult to search through all these databases and repositories to locate the right document and the up-to-date information they need. Because of this situation and because of the relative development independence site B has, some local codification strategies have developed and site B maintains its own internal way of documenting knowledge. This independence, but also separation, from the working methods of site NL, inevitably creates a gap in knowledge sharing between site NL and site B.

"*We have a SharePoint where we put a lot of the documentation; that's a recent technology we use here. I think it's been introduced over the last year. And it's convenient for all people because previously that was quite a mess, because you have one version of the document in one share folder on the network, and the other, the same document elsewhere.*" (Software Engineer, site B)

Documentation storing and sharing between site NL and site A also has certain characteristics. Because of the information barriers between them, the team at site A does not have access to all the documentation available in the databases of the NL site. As a consequence, site A employees need to ask their colleagues from the NL site for any extra information they might need. In turn, site NL people need to put extra effort and time to find that information and send it back to site A. In Océ, proactiveness is encouraged, but so is asking for information one needs to do a proper job.

"I don't mind. I'm actually happy that they are asking for information, because we've seen a lot of cases where they just didn't ask and they did something or assumed something, and afterwards they complained they didn't have anything - they didn't ask! We're really in a mode that people should ask for information, so I'm really happy they ask, so then already I quickly see what it's about and see if I can help" (Software Engineer, site NL)

C. Knowledge Transfer

The majority of the project is developed at the NL site and therefore most of the system-generic knowledge also resides in site NL. This inevitably causes knowledge to be "sticky" [8], which means that it takes additional effort for that knowledge to be transferred from the NL site to the remote sites.

"We don't feel the same handicaps as our remote colleagues, and maybe, I hope I do my best to keep them in touch and things like that, but yes, here we can act much faster." (System Integrator, site NL)

Another reason that enforces the stickiness of knowledge is the lack of the actual physical printing machines at the remote sites. The teams in the remote sites develop and test their work using simulators or they have a type of printer with limited functionality. This restricts their knowledge on how the entire system works, and their capabilities to perform as good as their colleagues in site NL.

"*But I do think that having a machine would help in understanding how the system works and, for now I don't understand yet why they don't have a machine.*" (Unit leader, site NL)

Concerning the unit-specific knowledge, the situation seems to differ between the co-development cooperation site NL-site A and the independent development site NL-site B. In the first case, function-specific knowledge remains in the NL site, and it is difficult to be transferred to the site A team. The main reason appears to be the lack of the printing machines in the remote site that limits their capabilities, as well as the unit-specific knowledge of the team in site A.

"*This year it was the first time that we saw also how this kind of testing is done. It was all taped and they made a sort summary and we just saw how something was tested and so on. But it was the first time we saw that. And then we heard the customer saying, yes, we'd like this and that and then we heard, yes, we will have to do that and that and you will get the requests and so on.*" (Tester, site A)

Another challenge in knowledge transfer is how to locate the knowledge. As we have previously observed, a strong personalization strategy is pursued in agile development methodologies. People rely more on the transfer of knowledge in a person-to-person way, and it becomes more efficient to know who knows what. Transactive memory is highly visible in the agile environment of the NL site and it has been recognized as an efficient way of improving performance, since people can find the right information by spending the least effort and time [17].

"And it makes it easier to know who is working on what part of the implementation, and that's also what we learn by being there, that you are more aware of what question should I ask and to whom. Don't just blow out your question and send it to five engineers, but when we talk about this part of the end phase, I know I have to contact this person" (Designer, site NL)

VI. Multi-Site Software Governance

A. Business Strategy

The contractual relationships and as a result the information barriers between site NL and site A seem to impact knowledge sharing between remote colleagues. People in site A do not have direct access to the documentation that is available in site NL, and as a consequence they rely on the selected information and documentation that site NL sends them.

"*Site A is some kind of different entity. And therefore, we always have to ask, please, or we don't have access to all the repository that they have and usually then we have to ask, please, could you please copy that from there to here so that we can have also access and sometimes it might be annoying but . . .*" (Software Engineer, site A)

Additionally, people in site NL and especially the software engineers have to invest more time and effort to filter the information and share it with their remote colleagues.

"*So they cannot access the Wiki, so in the beginning we put a lot of stuff in the Wiki and they couldn't access it, so problem. Later they said, well, we could use sharepoint and put stuff on there, and you have versioning so it's kind of a mix between Wiki and storing documents and, well, I've actually never really used it a lot.*" (Software Engineer, site NL)

Finally, the flat organization and the agile environment in site NL enforce the quick response time from people working in that site. Software engineers in site NL are encouraged to take responsibilities and initiative and act accordingly and they can quickly reply to their colleagues

in site A, increasing the speed of communication between the two remote locations.

"I expect that people are perhaps in meetings and they are quite busy, but usually they are able to answer me right away and its very, very good for us." (Software engineer, site A)

B. Team Structure and Composition

In distributed teams the lack of team cohesion is a recognized challenge [32]. Team members in remote locations are less likely to perceive themselves as part of the same team, compared to team members working in a co-located environment. Moreover, in a highly agile environment, there is less attention to documentation and as a result much knowledge remains tacit. Transferring tacit knowledge requires personal interactions and in a multi-site environment these interactions need more effort and time to take place.

As described earlier, the largest part of the project remains in the NL site. Consequently, more people are working there and more knowledge resides at that site. Most of the software units are developed in site NL, and most of the software unit-specific decisions are taken there. Additionally, the unit leaders and system integrators are located in site NL which means that also all the system-generic decisions are taken in site NL. Since most of the project knowledge is created in site NL and remains agile, people are less likely to understand the need to transfer that knowledge to their remote colleagues.

"*We are big brother, and they are only small team, and yes, we can settle a lot of issues, at the coffee machine, and yes, they don't participate. But that's the organization. If we were with three equally groups, one in site A with their responsibility, one in site B with same responsibility, and one in site NL with same responsibility, that would be different. We also would feel that we can't just decide something at the coffee machine, we have to put it on paper and we have to agree before the parties, but since we have the system architect and the biggest role, its too easy not to document things, too easy not to have a meeting about it, I think that's the case for a large part.*" (Unit leader, site NL)

The hierarchical organization of the teams might influence knowledge sharing. Previous research suggests that as the role differentiation increases in software development teams, it leads to a decrease in interaction and a corresponding decline in the shared mental model [33]. Within Océ, it is perceived that the hierarchical organization is more intense among the team structures in site B. By hierarchy, we mean that a unit leader for example has more decision making power on design and development matters than a developer. As a result, developers in site NL believe that their colleagues in site B need more time to take an action (e.g. fix a bug or work on a Change Request) and that often can cause delays; this has not been quantitatively confirmed though.

"*I had to wait for days through the official channel of delivering, of updating.*" (Software Engineer, site B)

"Usually I'm telling him I discussed this or if somebody is requesting to do something first we have to ask here on our site, is it okay for you if I do this because I was requested, and if he says, yes, then we are doing it. If he says, no, we have to say, sorry, can't do it yet." (Software Engineer, site A)

Finally, team structure and composition can impact the communication speed and frequency. Based on the way teams are organized within the company, interaction between remote locations is more likely to occur through the "contact persons". This can potentially create communication bottlenecks and delays.

"It's better if you inform your responsible before you contact. But it might depend on responsible, also. There are some that they need to know everything before you contact the remote location, and sometimes you have a responsible that delegates to you the responsibility of something." (Software Engineer, site A)

C. Task Allocation

In task allocation, we examine the way development activities are distributed among sites. We have already mentioned the two main task allocation methods identified in the Océ case study (co-development between site NL-site A and more independent development between site NL-site B). During the analysis, several knowledge challenges appear to be influenced by this construction. An important aspect is that, most often, the task allocation is tightly coupled with the team structure and composition [34], and more specifically in distributed development environments [35].

One of the first things to be noticed is the influence of task allocation on the communication frequency. As already described previously, communication frequency between site NL and site A is higher. On the other hand, communication frequency between site NL and site B depends more on the development phase. Consequently, we observe that tightly-coupled activities require a more intense communication compared to loosely-coupled activities. Team composition also plays a role in this case, as the fluctuation in the communication frequency refers primarily to software engineers and unit architects. Unit leaders who communicate for progress status and planning purposes have a more stable communication frequency based on scheduled regular meetings. For example, there is a Software Progress Committee that meets once every week to discuss project planning and current

status.

In addition, the codification strategy appears to be influenced by task allocation. In the case of co-development between site NL and site A, where development activities are tightly coupled, it is also noticeable that remote colleagues are more dependent on documentation sharing. Site B, however, has a self-sufficient domain knowledge and their communication with site NL is on interfacing and integration. There is a local Sharepoint site within site B and the team has its own, local procedures of how and when to document and share knowledge, limiting their codification dependencies with site NL.

The location where domain knowledge resides, is also connected with task allocation. In co-development, all knowledge (system-generic and unit-specific) remains in site NL and therefore additional effort needs to be invested in knowledge sharing between the remote sites.

Finally, task allocation can have a social impact on knowledge management. The motivation of site A colleagues can be influenced by the lack of responsibility in the development tasks that they receive, and they more often leave the company. When an employee leaves the company that means that he or she takes the knowledge obtained so far, and at the same time Océ has to hire someone else, train him and integrate him with the team, the project and the company.

"*You just get small pieces of the cake. This is also frustrating for the guys here, because they are, in terms of qualification, they know the technology, at least the same level as the guys in site NL, but they lack the top level domain knowledge, and this causes instability in the team, because one could say, okay, I'm very good in Java. I can do more than this small piece of user interface.* " (Team leader, site A)

VII. Lessons Learned

We have examined two multi-site software governance structures and their differences. The first difference relates to the business strategy that binds the relationship between the remote offices. Between site NL and site A there are information barriers. The next difference is the way tasks are allocated. The site NL and site A teams co-develop a software unit, which implies that the development activities and dependencies between the two sites are tightly coupled. On the other hand, site B is responsible for the development of a complete software unit, meaning that the development activities are more independent from site NL. Finally, the two governance structures are similar as far as team structure and team composition is concerned. Site NL is a flat organization, with small distances between roles, and agility is highly supported. The remote sites however are more hierarchically structured and focus more on strict processes and procedures.

Table II
The impact of Multi-site Software Governance on Knowledge Management

Multi-site Software Governance		Knowledge Management Challenges
Business Strategy	**Site NL-Site A:** • They are different companies and information barriers exist between the remote sites	• No direct documentation due to information barriers. • Information sent from Site NL to Site A needs to be filtered. • Communication frequency is higher.
	Site NL-Site B: • They are the same company and no information barriers exist between the remote sites	
Team Structure & Composition	**Site NL-Site A:** • Site NL is a flat organization, while Site A is hierarchically structured. • Role descriptions differ between sites. • Unbalanced team sizes.	• Hierarchical structures create bottlenecks in knowledge sharing. • Too much focus on agility stresses tacit communication and documentation remains outdated. • Different role descriptions makes knowledge difficult to locate. • Knowledge tends to stick where the majority of teams, or where the larger teams are located.
	Site NL-Site B: • Site NL is a flat organization, while Site B is hierarchically structured. • Role descriptions differ between sites. • Unbalanced team sizes.	
Task Allocation	**Site NL-Site A:** • They co-develop a function and their activities are tightly coupled.	• Tightly coupled activities increase the need for knowledge sharing. • Co-development creates a greater need for codified knowledge. • Communication frequency is high.
	Site NL-Site B: • They develop independently and their activities are loosely coupled.	• Knowledge tends to stick to the independent development teams. • Communication frequency depends on the release phase.

Having defined the two cases of multi-site software governance, we investigated the impact different governance structures might have on several knowledge management challenges. Table II presents our main findings. The knowledge management challenges identified within the case study concern the communication, the knowledge creation and storage and the knowledge transfer. The main impacts observed are:

- The information barriers between offices increase the effort and time spent on managing the creation, storage and sharing of knowledge, both tacit and explicit.
- The unbalanced structure and composition of teams impedes the smooth flow of knowledge. For example, when the team leader of the unit is located in the NL site, unit-specific knowledge "sticks" to the NL site.
- Allocating tightly-coupled activities among remote teams, increases the need for knowledge sharing and more effort needs to be spent in knowledge transfer. The communication frequency is higher because team members need to be in contact on an everyday basis for the coordination of their development activities.

Summarizing our observations, the software governance between site NL and site A has a stronger impact on knowledge management than the software governance between site NL and site B. Whether the observed impact is beneficial for the productivity and the performance of the

distributed project –for example in terms of development speed or number of residual bugs– needs further study. We round off this research with a structural perspective of multi-site software governance associated with *(a)* the business strategy between the remote offices, *(b)* the structure and composition of the distributed teams, and *(c)* the allocation of the development tasks across sites. We believe that these attributes should be considered for the success of a multi-site software governance structure. We should not forget, however, that there is no "one size fits all" solution to governance but that an effective governance is rather dependent on the situational characteristics of a software company [1].

VIII. Conclusion

We have defined a multi-site governance structure based on *business strategy*, *team structure and composition* and *task allocation*. We also identified several knowledge management processes, within the distributed software development environment, dealing with communication, knowledge creation, storage and sharing. In our case study, information barriers between sites, unbalanced team composition and tight coupling of distributed activities impact knowledge management processes.

Multi-site software governance should be further researched and enriched, by eliciting more information and identify other governance aspects besides we identified so far. By creating a more concrete definition of what constitutes multi-site software governance, we can build more accurate models and perform additional case studies on how to organize and administer multi-site software development activities at a strategic, tactical as well as operational level. Our aim is to expand and supplement the results of the present case study and create a multi-site governance framework based on best practices. This way we obtain a sound insight into what steps should be taken, and how organization and development activities should be structured to best align business and software development goals.

Acknowledgment

This research has been partially sponsored by the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) program on Software Engineering Research via contract 638.001.406 GRIFFIN: a GRId For inFormatIoN about architectural knowledge and the Dutch "Regeling Kenniswerkers", project KWR09164, Stephenson: Architecture knowledge sharing practices in software product lines for print systems.

References

[1] P. L. Bannerman, "Software development governance: A meta-management perspective," in *Proceedings of the 2009 ICSE Workshop on Software Development Governance*, ser. SDG '09. IEEE Computer Society, 2009, pp. 3–8.

[2] N. Ramasubbu and R. K. Balan, "Towards governance schemes for distributed software development projects," in *Proceedings of the 1st international workshop on Software development governance*, ser. SDG '08. New York, NY, USA: ACM, 2008, pp. 11–14.

[3] R. Heeks, S. Krishna, B. Nicholson, and S. Sahay, "Synching or sinking: Global software outsourcing relationships," *IEEE Software*, vol. 18, pp. 54–60, 2001.

[4] T. Dingsøyr, R. Conradi, and S. Telecom, "A survey of case studies of the use of knowledge management in software engineering," *International journal of software engineering and knowledge engineering*, vol. 12, no. 4, p. 391, 2002.

[5] A. Tiwana, "An empirical study of the effect of knowledge integration on software development performance," *Information and Software Technology*, vol. 46, no. 13, pp. 899 – 906, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/B6V0B-4CHGF41-2/2/74d374a68b357388303c799d0792377f

[6] S. L. Jarvenpaa and D. E. Leidner, "Communication and trust in global virtual teams," *Organization Science*, vol. 10, pp. 791–815, 1999. [Online]. Available: http://portal.acm.org/citation.cfm?id=767698.768296

[7] J. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," *IEEE Transactions on Software Engineering*, vol. 29, no. 6, 2003.

[8] G. Szulanski, "The process of knowledge transfer: A diachronic analysis of stickiness," *Organizational Behavior and Human Decision Processes*, vol. 82, no. 1, pp. 9 – 27, 2000.

[9] S. Chulani, C. Williams, and A. Yaeli, "Software development governance and its concerns," in *Proceedings of the 1st international workshop on Software development governance*, ser. SDG '08. ACM, 2008, pp. 3–6.

[10] H. Gewald and K. Helbig, "A governance model for managing outsourcing partnerships: A view from practice," in *Proceedings of the 39th Annual Hawaii International Conference on System Sciences - Volume 08*. IEEE Computer Society, 2006.

[11] E. Carmel and P. Tija, *Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce*. Cambridge University Press, 2006.

[12] H.-L. Yang and J.-H. Tang, "Team structure and team performance in is development: a social network perspective," *Information Management*, vol. 41, no. 3, pp. 335 – 349, 2004.

[13] M. B. O'Leary and J. N. Cummings, *The spatial, temporal, and configurational characteristics of geographic dispersion in teams*. MIS Quaterly, 2007, vol. 31, no. 3.

[14] J. A. Espinosa, S. A. Slaughter, R. E. Kraut, and J. D. Herbsleb, "Familiarity, complexity, and team performance in geographically distributed software development," *Organization Science*, vol. 18, no. 4, pp. 613–630, 2007.

[15] M. B. O'Leary and M. Mortensen, "Go (con)figure: Subgroups, imbalance, and isolates in geographically dispersed teams," *Organization Science*, vol. 21, no. 1, pp. 115–131, 2010.

[16] A. Kofman, A. Yaeli, T. Klinger, and P. Tarr, "Roles, rights, and responsibilities: Better governance through decision rights automation," in *Proceedings of the 2009 ICSE Workshop on Software Development Governance*, ser. SDG '09. IEEE Computer Society, 2009, pp. 9–14.

[17] S. Faraj and L. Sproull, "Coordinating expertise in software development teams," *Manage. Sci.*, vol. 46, no. 12, pp. 1554–1568, 2000.

[18] A. Lamersdorf, J. Munch, and D. Rombach, "A survey on the state of the practice in distributed software development: Criteria for task allocation," *Global Software Engineering, International Conference on*, vol. 0, pp. 41–50, 2009.

[19] A. Mockus and D. M. Weiss, "Globalization by chunking: A quantitative approach," *IEEE Softw.*, vol. 18, pp. 30–37, 2001. [Online]. Available: http://dx.doi.org/10.1109/52.914737

[20] P. R. Carlile, "Transferring, Translating, and Transforming: An Integrative Framework for Managing Knowledge across Boundaries," *Organization Science*, vol. 15, no. 5, pp. 555–568, 2004.

[21] P. J. Ågerfalk, B. Fitzgerald, H. Holmström, B. Lings, B. Lundell, and E. Conchúir, "A framework for considering opportunities and threats in ditstributed software development," in *International Workshop on Distributed Software Development*. Austrian Computer Society, 2005.

[22] D. E. Damian and D. Zowghi, "The impact of stakeholders? geographical distribution on managing requirements in a multi-site organization," in *RE '02: Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 319–330.

[23] F. F. Correia and A. Aguiar, "Software knowledge capture and acquisition: Tool support for agile settings," in *ICSEA '09: Proceedings of the 2009 Fourth International Conference on Software Engineering Advances*. IEEE Computer Society, 2009, pp. 542–547.

[24] J. D. Herbsleb and D. Moitra, "Guest editors' introduction: Global software development," *IEEE Software*, vol. 18, pp. 16–20, 2001.

[25] I. Oshri, P. Van Fenema, and J. Kotlarsky, "Knowledge transfer in globally distributed teams: the role of transactive memory," *Information Systems Journal*, vol. 18, no. 6, pp. 593–616, 2008.

[26] D. Wegner, *Transactive memory: A contemporary analysis of the group mind*. Springer-Verlag, 1987, pp. 185–208.

[27] J. Kotlarsky and I. Oshri, "Social ties, knowledge sharing and successful collaboration in globally distributed system development projects," *Eur. J. Inf. Syst.*, vol. 14, no. 1, pp. 37–48, 2005.

[28] A. L. Strauss and J. M. Corbin, *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications, Inc., 1998.

[29] R. K. Yin, *Case Study Research: Design and Methods*. Sage Publications, Inc, 2003.

[30] E. Carmel and R. Agarwal, "Tactical approaches for alleviating distance in global software development," *IEEE Softw.*, vol. 18, no. 2, pp. 22–29, 2001.

[31] *What's your strategy for managing knowledge*, vol. 77, 1999.

[32] B. Ramesh, L. Cao, K. Mohan, and P. Xu, "Can distributed software development be agile?" *Commun. ACM*, vol. 49, no. 10, pp. 41–46, 2006.

[33] L. L. Levesque, J. M. Wilson, and D. R. Wholey, "Cognitive divergence and shared mental models in software development project teams," *Journal of Organizational Behavior*, vol. 22, no. 2, 2001.

[34] A. Lamersdorf, J. Münch, and D. Rombach, "A decision model for supporting task allocation processes in global software development," in *Product-Focused Software Process Improvement*, ser. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2009, vol. 32, pp. 332–346.

[35] A. Barcus and G. Montibeller, "Supporting the allocation of software development work in distributed teams with multi-criteria decision analysis," *Omega*, vol. 36, no. 3, pp. 464 – 475, 2008.

[36] V. Clerc, P. Lago, and H. van Vliet, "Managing Architectural Knowledge in GSD: What to Share, and How to Do It!" 2011, submitted.