RECURRENT KNOWLEDGE BOUNDARIES IN OUTSOURCED SOFTWARE PROJECTS: A LONGITUDINAL STUDY

Complete Research Paper

Winkler, Maike, University of Bern, Switzerland, maike.winkler@iwi.unibe.ch

Brown, Carol, Stevens Institute of Technology, USA, carol.brown@stevens.edu

Huber, Thomas, University of Bern, Switzerland, thomas.huber@iwi.unibe.ch

Abstract

Knowledge boundaries can constrain cross-border collaboration. Based on a qualitative case study of a distributed team, we examine which semantic knowledge boundaries recur and why they recur over the life of an agile, outsourced software project. Based on our analysis of observational data, collaboration tool data and interviews, we first identify the similar recurrent boundaries and categorize them under three domains for this type of software application: Assembling, Designing, and Intended user interaction. We then examine three practices utilizing software prototypes that team members used to bridge them. First, we find that similar semantic knowledge boundaries related to all three of the domains recur over the 10-month life of the project. Second, we find that team members repeatedly enact the same practices to bridge similar knowledge boundaries over the life of the project. Our data also suggest that team members in outsourced agile software development projects will likely use practices to transcend, rather than traverse, knowledge boundaries. Based on these emergent findings, we develop three propositions for future testing. Our study therefore contributes to the growing research streams on knowledge boundaries in IS outsourcing and the usage of prototypes in agile software development.

Keywords: Knowledge boundaries, team collaboration, outsourced workforce and governance, IS outsourcing, agile development methods, object bridging practice

1 Introduction

It is well known that knowledge differences can impede cross-border team collaboration by creating knowledge boundaries (e.g., Majchrzak et al., 2012, Carlile, 2002). Heterogeneous knowledge bases and specializations cause different "thought worlds" (Dougherty, 1992). Consequently, team members define, and perceive situations from various perspectives (Baba et al., 2004, Boland and Tenkasi, 1995) and do not interpret things in the same way (Cramton, 2001, Bechky, 2003, Carlile, 2002) – that is, they encounter semantic knowledge boundaries (Carlile, 2002). Acquiring an understanding is crucial for team members to effectively collaborate across organizational borders and has been acknowledged as a major concern in the dispersed team literature (e.g., Cramton, 2001, Jarvenpaa and Leidner, 1998), particularly in the offshore information systems development (ISD) literature (e.g., Levina and Vaast, 2008, Majchrzak et al., 2005, Sarker and Sahay, 2004, Carmel and Tjia, 2005).

In the offshore ISD context, client and vendor often have extreme knowledge asymmetries (Vlaar et al., 2008). The client has business knowledge to develop the ideas for the software whereas the vendor employees have relevant technical knowledge (Tiwana, 2004). The resulting different perspectives and conceptions make it particularly challenging to interpret requirements (e.g., Bergman et al., 2002, Gorschek and Wohlin, 2006). For example, vendor employees are likely to "inconsistently ambiguously, and inaccurately interpret client needs" (Tiwana, 2004, p. 6). Hence, issues may endure over a longer

period of time (Herbsleb and Moitra, 2001). The earlier interpretive differences are revealed, the faster they can be resolved by team members; otherwise, it can be costly and difficult to resolve them (Rowen, 1990). Since client and vendor employees are geographically distributed (Dibbern et al., 2008), advancing our understanding of practices to resolve differences in the interpretation of requirements needs much closer investigation (Damian and Zowghi, 2003, Hull et al., 2010).

With the rise of agile software development practices with short iteration cycles and rapid prototyping at its core (Schwaber and Beedle, 2002), the software prototype has become a common object facilitating learning between team members (e.g., Highsmith, 2013, Cao and Ramesh, 2007). Immediate feedback based on new revised prototypes may enable software developers to resolve differences in interpretations more quickly. Yet, while it is well acknowledged that bridging knowledge boundaries in offshore-outsourced projects represents a major challenge (e.g., Dibbern et al., 2008, Levina and Vaast, 2008, Kotlarsky et al., 2014), little is known about how practices with software prototypes may help to solve interpretive differences within an agile software development project over the life of a project.

Therefore, in this study we seek to address the following general research questions: *Which knowledge boundaries recur over the life of an outsourced agile software development project when software prototypes are used and why do they recur?*

The paper is structured as follows. First, we present some relevant literature for our qualitative research study, and elaborate on our research design. Then we present the findings that emerge from our qualitative analyses, from which we develop propositions for future research. Finally, we summarize our contributions and the implications for researchers and practitioners related to recurrent interpretive differences when software prototypes are used in outsourced software projects.

2 Background

2.1 Knowledge sharing approaches

Scholars agree that knowledge sharing is important to solve interpretive differences. However, there are competing perspectives with regards to how this knowledge sharing occurs and how much knowledge needs be shared.

On the one hand, scholars suggest that team members need to engage in rather effortful and time-consuming practices to *traverse* knowledge boundaries to converge knowledge bases and acquire a common understanding across organizational borders (e.g., Carlile, 2002, Bechky, 2003). Traversing requires team members to "identify," "elaborate" and "then explicitly confront the differences and dependencies" across boundaries through negotiations. This requires deep dialogue, time and resources (Majchrzak et al., 2012 p. 951). Thus, vendor employees need to acquire in-depth business knowledge and the client needs to obtain in-depth technical knowledge (Tiwana, 2004). In the ISD context, traversing practices can be perceived as deeper knowledge sharing practices between client and vendor that help them to gain a comprehensive understanding, i.e. an understanding that goes beyond the specific task at hand.

On the other hand, another stream of literature suggests that team members can integrate just enough knowledge to *transcend* knowledge boundaries without acquiring an in-depth, shared understanding to collaborate (e.g., Faraj and Xiao, 2006, Schmickl and Kieser, 2008, Majchrzak et al., 2012, Kellogg et al., 2006). That is, team members tend to "explicitly avoid boundary distinctions through minimizing differences and distinctions between specialty areas" (Majchrzak et al., 2012). This approach prevents the risk of "becoming trapped in recommending practitioners to engage in costly and time-consuming activities such as education and training to equalize their knowledge and experiences" (Vlaar et al., 2008, p. 232). In the ISD context, transcending practices can be perceived as sufficient knowledge sharing practices between client and vendor that help them to acquire a sufficient understanding. Software developers aim to gain just enough knowledge to complete one specific task instead of a comprehensive understanding of the overall, "bigger picture" of the software project.

Further, scholars across the fields of Sociology, Management and Information Systems have proposed the use of *boundary objects* in these knowledge sharing approaches to facilitate cross-border collaboration (e.g., Carlile, 2002, Carlile, 2004, Majchrzak et al., 2012, Kellogg et al., 2006, Star and Griesemer, 1989, Levina and Vaast, 2005, Bechky, 2003). Practices involving a variety of boundary objects have recently received more attention by IS scholars. Different artifacts such as sketches and design drawings or prototypes can be used as boundary objects (e.g., Star and Griesemer, 1989, Henderson, 1991, Bechky, 2003, Carlile, 2002). These types of objects serve as reference points, while also being flexible enough to adapt to the needs of each organizational site (Star and Griesemer, 1989).

Next we describe how the usage of prototypes as boundary objects in agile software development projects enables the client and vendor to bridge knowledge boundaries.

2.2 Using software prototypes for learning in agile software development

In agile software development contexts, client and vendor can use software prototypes to bridge knowledge boundaries. During weekly planning and review meetings they present the software prototype to elicit knowledge about requirement and discuss their work, goals and obstacles (Danait, 2005). Thus, the prototype has become a common object facilitating interactions between team members even in offshored settings. However, the software prototype does not automatically function as boundary object (Star, 2010, Levina and Vaast, 2005); the ability to operate as boundary object depends on how the prototype is used in practice (Nicolini et al., 2012, Seidel and O'Mahony, 2014, Majchrzak et al., 2012). Depending on how the software prototype is used in software development processes it may facilitate cross-border learning.

Scholars describe software development as "a mutual learning process" between the involved stakeholders (Kautz et al., 1992, p. 51) whereby learning is defined as the "growth in knowledge." (Sørensen, 2009, p.130) Yet, traditional software development methods that have separated the software development process into different phases discourage such learning (Kautz et al., 1992), particularly since feedback is delayed over longer periods of time (Cao and Ramesh, 2007). In contrast, scholars suggest that agile software development facilitates learning during the development process (e.g., Dyba and Dingsoyr, 2009, Nerur and Balijepally, 2007, Cao and Ramesh, 2007, Highsmith, 2013, Meso and Jain, 2006). The short iterative cycles (Awad, 2005) and frequent interactions that are part of agile software development methods result in immediate feedback, which promotes learning across team members (Meso and Jain, 2006, Cao and Ramesh, 2007, Berczuk, 2007). Frequent revisions of software prototypes also support iterative learning (Kruchten, 2001): As team members are able to reflect on what has (not) worked after iterations, they can adjust and change the process, practices and artifacts to make improvements (Kruchten, 2001, Dingsøyr et al., 2012p. 1214, Highsmith, 2013). Thus, given that frequent and immediate feedback on prototypes is a common practice, one would expect the recurrence of knowledge boundaries to diminish over the life of an agile software development project.

2.3 Study Background

This paper is part of a larger study in which we investigate how knowledge boundaries can be bridged in an outsourced software project when using the software prototype as boundary object (Winkler et al., 2014). Our initial focus was on examining how different object bridging practices involving a software prototype are used to bridge three different types of knowledge boundaries: syntactic, semantic, and pragmatic (Carlile, 2002). We also distinguished between two types of software prototypes that team members use in their boundary spanning practices as part of their agile software development practices: 1) the *building prototype (BP)*, the more hypothetical system of the prototype that needs to be erected to carry out and plan the software development task (e.g., written software requirements, design drawings) and 2) the *working prototype (WP)*, the system of the prototype that has already been developed and is thus represented through functioning software prototype (adapted from Lyytinen and Newman,

2008). By focusing on the boundary spanning practices for what we refer to as an episode – i.e., identifying the occurrence of a knowledge boundary and practices used by the team members that involved one or both types of prototypes to bridge the three different types of knowledge boundaries – we identified five distinct use practices (UP) being utilized over the first six months of the project.

Surprisingly, during our on-going observations of virtual meetings between client and vendor for this study, we also noted that over time similar semantic boundaries recurred, i.e., it seemed as if there were instantiations of similar issues that continued to occur. That is, the outsourced software developers seemed to continue interpreting similar project requirements differently than intended by the client over the project's life. Thus, the primary motivation for the study reported in this paper is to investigate this unanticipated phenomenon – i.e., which semantic knowledge boundaries recurred despite the usage of bridging practices that involved software prototypes to resolve them at an earlier point in time, and why.

The following three practices associated with semantic knowledge boundaries that were reported previously (Winkler et al., 2014) are therefore of primary interest here.

Contrasting: Visually contrasting the BP with the WP involved activities where the team members contrasted the BP with the WP by switching between screen shots, tables, and user story descriptions and contrasting them with an already implemented functionality, feature or visual design. For example, a developer opened the current prototype (WP) displaying a design that has already been implemented. In addition, he opened a not yet implemented screen shot (BP) to contrast it with the already implemented design to identify similarities and differences.

Exemplifying: Visually exemplifying in the BP or WP involved activities such as moving the mouse cursor to the relevant part of either the BP or WP to visually highlight the term or click-through scenarios. For example, a developer opened the current working prototype. The developer moved the mouse cursor to a specific working functionality and clicks through an example.

Relating: This use practice, which was only applied by the client, involved orally relating a concept in the WP or BP to a more holistic view in combination with Contrasting or Exemplifying, such as explaining the reason and meaning behind functionalities and how they are connected to the overall idea for the software application. For example, the client used a table (BP) with general descriptions that are relevant to different requirements to elaborate on how an idea in one requirement relates to the overall software.

As will be described in more detail in our Methods section below, for this phenomenon-driven study we also expanded our case study scope to include episodes over a 10-month timeline.

3 Methods

3.1 Case Selection, Data Collection and Empirical Context

Due to the lack of in-depth field studies on using software prototypes as boundary objects, our research approach involves collecting and analyzing data from a single case in-depth (Yin, 2009, Sarker et al., 2012). We observed something unexpected: similar differences in interpretations between the client and vendor seemed to re-occur. For this particular study, our research was therefore guided by a phenomenon-based approach aiming to capture, portray and conceptualize a phenomenon (von Krogh et al., 2012, p. 278) that emerged from our initial analysis of the Case study.

Our in-depth case study data was collected over a 10-month offshore-outsourced software development project that was being conducted using agile software methods. A client situated in Switzerland who had an idea for a new software application decided to offshore the software development project to Vietnam. A small offshore team located in Vietnam (6 team members) was responsible for developing a novel software tool using agile development methodologies which we refer to as TechProduct. The

client was responsible for the creation of software requirements (i.e., user stories) which were then assigned to software developers in Vietnam. The software developers and Scrum Masters who facilitated communication between them and the client had daily internal meetings where they discussed ideas and problems with each other. Jointly they collected any open questions as a team to subsequently ask the client. When a developer started working on a particular user story, he presented and discussed his work-in-progress or completed work in virtual meetings taking place 2-3 times a week. In those meetings, developers and client shared and used the current software prototypes (BP and/or WP) via screen sharing.

The vision for the initial version of TechProduct was to provide users with a single social media tool which enables them to manage information in various ways. Each user can create Tecs which contain compressed information and share it with other users. This context and setting was appropriate for our research purpose since team members engaged in activities with the software prototype in agile software development contexts. Prototyping took place in an iterative manner throughout the software development process. Since the knowledge bases of the client and the offshore software developers were very different, interpretive differences were likely to remain throughout the project. That is, the client developed the software idea but did not have any experience with software development. Whereas the software developers were not involved in the development of the software idea, they had the technical knowledge to implement the requirements. In addition, the level of interdependence between the client and developers was high: developers had to work with the outputs of the client (e.g., design sketches or user stories as part of the BP) to understand requirements, whereas the client relied on the outputs of the developers (the WP) who implemented these requirements. Thus, in the weekly meetings, team members repeatedly engaged in practices with the software prototypes to address boundaries that occurred.

During the first six months of the project, team members focused on the implementation of the basic TechProduct functionality and the design of the user interface, Tecs and profiles. The WP at the end of this time period enabled multiple users to store and share information via Tecs with each other. During the four subsequent months, the team focused on the implementation of further functionalities and features, as well as the initial implementations on administrative support features (frontend and backend); for example, the client was able to manage user queries. This and other features were also tested in a closed beta test during this time period, and this user feedback led to an increased number of requirements to make improvements to basic functionalities and design. This 10-month time period therefore enabled us to examine what is similar about semantic boundaries and in what ways the software prototypes is used to bridge them over a project's life.

To investigate our research questions for this study, we collected process data from the same software development project team members. To reconstruct events and the time of events, we triangulated process data from three different sources. Process data is comprised of "stories about what happened and who did what when – that is, events, activities, and choices ordered over time" (Langley, 1999 p. 692).

(1) Recorded observational data: We attended and recorded all virtual (video and audio) meetings between the software developers and the client. These meetings took place between 2-3 times a week over the 10-month period. During these interactions, the developers in Vietnam shared the latest prototype (WP and/or BP) and discussed changes, problems and achievements with the client. In total we attended 91 meetings (average time: 23.5 min) resulting in 35.7 h and 339 pages of notes. The actual recordings allowed us to assess when team members faced boundaries when working on different requirements over a software development project as well as the practices with software prototypes the team members engaged in.

(2) Collaboration tool data: The team used a collaboration tool called Assembla to organize and coordinate software development processes. The tool encompassed all requirements and provided an overview to everyone in the team to raise awareness on who is working on what task. All log files and content exchanged via Assembla during the 10-month time period were tracked. This allowed us for instance to triangulate data of specific observations with additional information.

(3) In-depth interviews: In addition to the above data sources, we conducted and transcribed a total of 12 interviews with individuals in two interview rounds via Skype. These interviews were semi-structured ranging from 30-90 min to allow an appropriate degree of flexibility regarding the order of questions and scope for answers (Opie and Sikes, 2004). In the first round of interviews, we conducted interviews with all project participants including the client (C), three developers (D1-3), two Scrum Masters (SM1-2) and the third-party designer (TPD). Interviews with two of the developers took place in form of a written survey due to language barriers. We asked about the project in general (e.g., complexity, aims etc.), the collaboration and communication across boundaries and asked them to elaborate on interpretive differences that they faced up to this point of the project. In the second round of interviews, we conducted interviews with five of the seven team members (e.g., client, developers, Scrum Master) that were still actively involved in the project and participating in virtual meetings. For these interviews we reviewed our notes from the virtual meetings as a basis for developing interview questions that focused on any changes within the project (e.g., tasks, processes) and the reasons for why changes occurred or did not occur. Then, we focused on how team members used the software prototypes to address boundaries at different points in time from the client's and the developers' point of view.

3.2 Data analysis

Our analyses for this study began at the episode level – i.e., identifying when a semantic boundary occurred and the specific boundary spanning use practice that was enacted. We then constructed a timeline so that we could explicitly examine the episodes involving a semantic boundary that occurred over the 10-month period. To answer the first part of our research question, *which semantic knowledge boundaries recur*, we used an open coding process (Corbin and Strauss, 1990) that enabled us to identify "similar" semantic boundaries and develop a categorization scheme. Our initial coding of the episodes that involved semantic boundaries resulted in six concepts: Color, Sequence, Form, Count, Display and Use (see Table 1). After comparing the episodes for each of these concepts we abstracted three higher-level categories (Corbin and Strauss, 1990) which we refer to as domains: Design (Color, Form), Assembling (Sequence, Display), and Intended user interaction (Use, Count). After coding all of the episodes by domain, we then determined whether and when a similar form of semantic boundary occurred over the 10-month project.

To answer the second part of our research question, *why semantic knowledge boundaries recur*, we then compared the practices with software prototypes (Contrasting, Exemplifying, Relating) that the client and software developers enacted for the recurring knowledge boundaries for each of the three domains. We then utilized our interview data to triangulate our findings and better understand the reasons given by the team members for why they engaged in certain practices.

In the next section we present our emergent findings and develop propositions that reflect our interpretations.

4 Findings

4.1 Similar semantic knowledge boundaries that recur

Our findings reveal that team members faced similar semantic knowledge boundaries related to three different domains over the life of the outsourcing project as they utilized agile software developments to create a new interactive online tool for users. More specifically, our analyses identified six concepts related to the specific TechProject, which we then categorized into one of three domains (Table 1). Appendix 1 provides examples for similar semantic boundaries that recurred first for the two concepts of each domain. Appendix 2 provides a detailed example of recurrent semantic boundaries for one of the domains (Design).

Domains	Concepts from TechProject	Example boundary	Coding Extract
Assembling: Interpretive differences on how information should be arranged	**Sequence:** Information should be presented in a certain order.	A team member did not understand in which order selected Tecs should be arranged	*"For unselected Tecs the order is kept, but what about selected?"* (Observation, Month 6)
	Display: Information should be seen from certain points of view.	A team member did not understand what Tecs and WebTecs of profile C can be seen when they are linked to profile C, but not to profile A or B	*"For Tec and WebTec we should see the content of profile C but I don't know if the visibility for this Tec in private or guide mode, i.e., what we can see"* (Observation, Month 7)
Designing: Interpretive differences on how text and visual representations of information are designed	**Color:** Texts or visual representations should be colored in a way that conveys certain meaning.	A team member did not understand which color Tecs should have when they are linked to a user's profile	*"Should the color be green?"* (Observation, Month 3)
	Form: Texts or visual representations should be formed in a certain way.	A team member did not understand whether there should be icons to link Tecs to a user's profile	*"Is a pin icon needed here?"* (Observation, Month 2)
Indented user interaction: Interpretive differences on how users should interact with the product's component	**Use:** Functionalities and feature should be used in a certain way	A team member did not understand what users can do with Tec groups	*"What can users do with Tec groups?"* (Observation, Month 6)
	Count: Instances of information sharing should be tracked	A team member did not understand why the Tec count should increase	*"We understand the use-count should go up, yes we do not see why?"* (Observation, Month 2)

Table 1. Categorization of semantic boundaries into three domains

Table 2 summarizes our findings in terms of the total number of similar semantic boundaries that occurred for each of the three problem domains over the life of the project. These counts demonstrate that similar semantic boundaries continued to occur at various times of the project for all three of these domains.

Month	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Total per domain**
Assembling	-	xxx	-	xx	xxxxxxx	xxx	xxxxx	-	-	xx	**22**
Designing	-	xx	xxx	-	x	-	xxxxxxx	-	xxxx	-	**17**
Intended user interaction	x	xxx	-	xxx	x	x	xx	-	x	x	**13**
Total per month	1	8	3	5	9	4	14	0	5	3	52

Table 2. Counts of similar semantic boundaries for three domains over 10-month project; (x): occurrence of similar semantic boundary; (-): no similar semantic boundary occurred within that month

In formal terms, our findings therefore suggest the following:

Proposition 1: When agile software methods are utilized for an offshore-outsourced software project, similar semantic knowledge boundaries related to multiple domains (assembling, designing and intended user interaction) will recur over the life of the project.

Next we look at the specific practices with software prototypes that were utilized when similar semantic boundaries occurred for each these domains.

4.2 Practices with software prototypes in agile software development

To address our research question about why boundaries recur, we examine what practices were utilized and why team members engaged in these practices. As found in the prior study, three different bridging practices were used when they faced semantic boundaries: Contrasting (C), Exemplifying (E), and Relating (R). In addition, however, our unexpected finding was that the team members continued to use the same use practice when boundaries related to a given domain recurred: *assembling, designing or intended user interaction*. That is, our overall finding is that similar practices were utilized when similar semantic knowledge boundaries were encountered (Table 3).

	Month									
Domain	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Assembling	-	C-BP&WP E-BP E-BP	-	R-BP E-WP	C-BP&WP E-WP C-BP&WP C-BP&WP E-BP E-BP C-BP&WP	C-BP&WP C-BP&WP C-BP&WP	C-BP&WP E-WP C-BP&WP E-WP E-BP	-	-	E-WP C-BP&WP
Design	-	R-BP &E-BP E-BP	E-WP E-BP E-WP	-	E-BP	-	C-BP&WP E-WP C-BP&WP &R-BP E-BP&R-BP C-BP&WP E-WP C-BP&WP	-	E-BP E-BP C-BP&WP E-BP	-
Intended User Interaction	E-WP & R-WP	R-BP E-WP&R-WP R-WP	-	E-WP C-BP&WP R-WP&C-BP&WP	E-WP	C-BP&WP & R-WP	E-WP E-WP & R-WP	-	E-BP	E-BP

Table 3. Practices for each domain over 10-month project, Contrasting (C), Exemplifying (E), and Relating (R); Building Prototype (BP); Working Prototype (WP)

More specifically, when similar semantic boundaries related to the *Assembling* domain were encountered, team members were more likely to engage in Contrasting BP & WP and Exemplifying in the BP or WP. For example, when a team member did not understand how the order of pages of a Tec should be arranged when a page that is linked to a user's profile is deleted. The team member showed the current prototype (WP) displaying what was implemented so far. The client recommended opening another requirement (BP). They opened the user story which described how a page of a Tec is deleted. They then contrasted the current prototype (WP) with a requirement (BP) that described how a page of a Tec is deleted to compare it to what is displayed in the current implementation. Then the client explained in an example in the WP: when there are 13 pages of a Tec and 1 page is deleted, other pages will move up depending on the time when the page was built.

Further, team members were more likely to engage primarily in Exemplifying in the BP or WP and Contrasting BP & WP which were infrequently combined with Relating in the BP when facing semantic boundaries related to the *Designing* domain. For example, a team member did not understand under which circumstances connecting lines have different colors. The developer opened screen shots attached to the user story displaying many colorful connecting lines. The client used the visual representation of the screen shot to indicate that there are three possible colors and discusses examples: blue, red and green lines, each indicating the level of ownership of Tecs. For instance, Tecs connected by a blue line belong to a user and the user can control the Tecs. He then related the idea to the whole by suggesting that the developer should *"close his eyes and imagine that in the future users can activate or deactivate a color."* The user can see immediately if there is *"a road for other users leading to my Tec here? It will improve the feeling about surfing here in Tec.com."* The developer then opened the current WP to contrast a similar already implemented functionality displaying a single connecting line (without color) with a screen shot of the not yet implemented colorful connecting lines.

Finally, when team members face similar semantic boundaries related to the *Intended user interaction* domain, they typically engaged in Exemplifying in the WP and Contrasting BP & WP which were often combined with Relating in the WP. For example, a team member did not understand what users can do with a functionality called "Tec groups." The Scrum Master showed and highlighted a written question (BP). Then he opened the current prototype (WP) displaying a Tec group in the background to contrast the question with an already implemented function. The user could enter the Tec group name and select a language in the current WP. The Scrum Master located the position in the prototype where users can create a Tec group. Then, the client orally related the concept to the whole. Tec groups are very important in the system enabling users to meet others who are also interested in a particular topic. He stated: *"As an architect I want to build a house, so I can create a Tec group for that house and invite only the workers – and share files with them."* Then he also explained that there will be Tec groups where a user can decide who can join in a Tec group. A user can create a Tec group of FC Basel and invite friends that can link to the Tec group. In the current system everyone can join a Tec group.

Proposition 2: When agile software methods are utilized for an offshore-outsourced software project, team members will likely enact the same use practices with software prototypes to bridge similar semantic knowledge boundaries over the life of the project.

In addition, our data reveal that the client and the vendor did not invest in deeply sharing knowledge. Instead, the team members shared just enough knowledge to quickly address a semantic boundary at hand. Our analyses of the interview data that we collected during the project suggest there were three primary reasons why.

First, software developers worked on one requirement at a time. To complete the requirement, they learned the information that was necessary to complete the single task at hand instead of gaining a more comprehensive picture of the whole software product. The client selectively elaborated on the overall software idea. As a result, it was difficult for vendor employees to understand the whole overall picture of Tec.com.

"*There are misunderstandings that occur... I am aware that the programmer now does something somewhere...one can already see it... for some reason some Tecs are green in the right column. If one had understood the whole project sincerely, then one shakes the head and says -a moment. And then it becomes clear that they have not yet fully understood it. But this is actually not a big difficulty. […] I think when one sits directly in the same room then one would discuss a lot about the big picture. This could have the advantage that all participants have a much better understanding where this journey is going and why something functions the way it does. This would of course also require time […] Would one bring all team members to the same level of knowledge one must somehow take three months to elaborate on it. And that is simply impossible. And that is why they simply have to trust me, the people there that I know what I am doing."* [C]

Second, although vendor employees had daily internal meetings in Vietnam, they did not attempt to deeply involve the client in knowledge sharing. During the meetings with the client they tried to share just sufficient knowledge and focus on delivering results. For instance, they did not share detailed technical information with the client that would take a lot of time. Instead, they focused on delivering the revised prototype as fast as possible. They also needed additional time to learn novel and modern technologies that were used in the TechProject, but that they were not familiar with.

"We don't try to talk much about, too much about detail, what we will do, so, to talk about what he need and what he expects, and we try to deliver." [SM1]

Third, the software prototype helped client and team members to convey their ideas and problems during cross-border interactions quickly. In particular, the use of prototypes helped to visualize ideas, examples and scenarios via screen sharing during virtual meetings. The client had a strong interest in realizing his ideas fast and aimed to accelerate the development process. The prototype helped him to transform his abstract ideas into something tangible; developing a prototype with working core functionality as fast as possible was important. The software prototype was a vital communication device to convey knowledge quickly.

"*[Screen shots] are simply a visual medium and it is simply clear, it is the fastest way. So, when you ask why, I could also write an A4 page but then I would have to write: Once you click on this bit and at the top left etc. So, this in words is …, so I am simply faster, when I rapidly take a picture and then draw circles in Photoshop with a pencil and make and arrow and keyword and then the issue is clear. Meanwhile, there are even tickets where there is simply a screenshot and not even some describing text.* [C]

Our findings therefore provide support for the observation previously reported by Majchrzak et al. (2012). That is, team members may seek to *transcend,* rather than traverse, knowledge boundaries. Stated more formally:

Proposition 3: When agile software methods are utilized for an offshore-outsourced software project, team members will likely enact use practices with software prototypes to transcend, rather than traverse, semantic knowledge boundaries over the life of the project.

5 Contribution and Implications

Our objective for this study was to investigate a phenomenon that emerged from our initial observations of the virtual team meetings for an outsourced software development project: i.e., *the recurrence of similar semantic knowledge boundaries over the life of an outsourced agile software project when software prototypes are used.* While previous outsourcing scholars have acknowledged the importance of bridging knowledge boundaries (Tiwana, 2004, Dibbern et al., 2008, Levina and Vaast, 2008, Vlaar et al., 2008), what has not been addressed in the existing literature was how practices with software prototypes are used to resolve interpretive differences over time. To the best of our knowledge this study is one of the first to explore this "temporal dimension of IT-mediated team behaviors (Shen et al., 2014)." Taking a phenomenon-based approach to analyze our rich, in-depth case study data enabled us to take into account complex, interrelated occurrences over time (von Krogh et al., 2012) and develop three propositions for future research.

Our first proposition is based on the emergent finding that similar semantic knowledge boundaries related to multiple domains *(Assembling, Designing and Intended user interaction)* will recur over the life of the project. This finding enriches the semantic boundary concept in the agile software development context. We provide a more fine-grained and contextualized coding example for each of these domains specific to the Techproject. However, we believe that our categorization approach will be applicable for other software projects, and that our first proposition can therefore be tested in similar and different project contexts.

Our second proposition is based on our observation that team members using agile software methods will enact the same practices with one or two software prototypes types (BP, WP) over the life of a project when they encounter similar semantic knowledge boundaries. This finding is consonant with previous findings where use practices were defined as "recurrent, materially bounded and situated action." (Orlikowski, 2002 p. 256) Our results provide evidence that team members use the software prototype as a "learning vehicle" (Floyd, 1984) to visualize thoughts, ideas or difficulties. That is, our results also suggest that team members preferred to learn by specific examples and comparisons by engaging in practices of exemplifying and contrasting. Software prototypes helped them to visualize specific scenarios and examples for one particular requirement. As cautioned by Naur (1984 p. 290), the effectiveness of a learn-by-example approach depends on "making the right generalization from the special cases shown in the examples." The team members found it challenging to generalize from a single requirement to the broader, overall idea of the software product. While focusing on specific examples helped team members to quickly bridge a particular boundary at hand, they continued to have difficulties applying knowledge to other requirements in the long run when they faced similar semantic boundaries. The practices transformed the software prototype as a boundary object into a "partial and temporary bridge" (Trompette and Vinck, 2009, Yakura, 2002) and enabled "speed and learning in the short term" (Kellogg et al., 2006 p.42). The team members engaged in activities that resulted in "intermediary scaffolds" to quickly create new versions of the artifact (Majchrzak et al., 2012). Thus, it is important to consider how the artifact is used in practice (e.g., Levina and Vaast, 2005, Nicolini et al., 2012, Seidel and O'Mahony, 2014, Barrett and Oborn, 2010).

Our findings therefore suggest that the reliance on agile software methods for an offshored custom development project will likely result in use practices for *transcending*, rather than traversing semantic knowledge boundaries. Our third proposition therefore also provides initial support for the assumption by Majchrzak et al. (2012) that continuously evolving, incomplete, fluid objects (such as software prototypes) support transcending practices. Thus, team members across boundaries share just sufficient knowledge to be able to collaborate without a synchronized, common understanding (Majchrzak et al., 2012, Kellogg et al., 2006, Faraj and Xiao, 2006, Schmickl and Kieser, 2008). In particular, our findings provide new insights on the concept of transcending (Majchrzak et al., 2012) in the context of agile software development. That is – while the engagement in transcending practices helped team members to quickly develop the software prototype in the short-term, it also allowed similar boundaries to recur in the long term. On the other hand, this emergent finding contradicts prior researchers that have argued that agile software development practices with frequent interactions and immediate feedback facilitate learning (e.g., Highsmith, 2013, Cao and Ramesh, 2007). If interpretive differences are identified and solved after immediate feedback, similar boundaries would not recur over the project's life. In our study, we indicate that similar boundaries can re-occur even in agile software development.

We believe our findings also provide valuable guidance for practitioners. First, our study provides strong evidence that team members can continually be challenged by interpretive differences over the life of an outsourced project using agile software development methods, and that team members are likely to enact similar practices to address them. Our categorizations of semantic knowledge domains and use practices that involve prototypes could be used to identify what boundaries are recurring and why. Managing risks in offshore development projects is necessary to achieve the potential gains of outsourcing and include risks resulting from different interpretations and perceptions (Kliem, 2004).

Second, our finding suggest that in order to take management actions, it is important to consider the trade-offs when using software prototypes in practice. When focusing on sharing "just enough" knowledge, team members may solve a boundary at hand quickly; yet, engaging in deeper knowledge sharing may help team members to ultimately identify underlying causes of such boundaries. Since agile work practices require fast responses to meet tight schedules and to frequently present work-in-progress to the client (Fowler and Highsmith, 2001), other management actions may need to be taken if sharing "just enough" knowledge for each requirement separately for a new outsourced IS application is a short-term approach that will not be valued for a long-term client-vendor arrangement.

5.1 Limitations and future research

Our study has several limitations. First, our findings are based on a single case study of an outsourced software project with a single client that was geographically distributed from the developers. Further, the software project involved developing a certain type of application: an online-based, interactive user application. While our triangulated, longitudinal data set and the internal replication opportunities allowed us to gain valuable insights from a single case study (Langley, 1999) and to develop three propositions, future studies are needed to test our propositions with data from other agile development projects, as well as in non-agile project contexts, as well as in other offshore contexts and domestic contexts.

Further, we reveal that similar interpretive differences continuously challenged the team members. While software prototypes helped to address immediate interpretive differences, "solving" recurrent problems such that similar semantic boundaries do not recur may require different practices with software prototypes. Thus, future research is needed to improve our understanding about how software prototypes need to be used to bridge interpretive differences in the short-and long-run.

In this study, we found that similar interpretive differences recurred and that team members used software prototypes to address them. However, future research is needed to investigate whether repeatedly bridging semantic boundaries with the same use practices that yield "temporary bridges" is more or less beneficial than engaging in practices for *traversing* semantic boundaries for a given IS outsourcing project. In addition, other contexts need to be investigated—including those in which team members expect to be engaged in a long-term client-vendor relationship and therefore seek to engage in deeper knowledge sharing practices.

Appendix 1: Similar semantic boundaries for 3 domains

Assembling		**Designing**		**Intended user interaction**	
Sequence	**Display**	**Color**	**Form**	**Use**	**Count**
A team member did not understand…					
…how the order of pages of a Tec changes when a page that is linked to a user's profile is deleted (Month 2)	…what a user can see when clicking on Tecs and profiles in a column (Month 2)	…what the design looks like when a Tec is colored as "shadowed" and if this applies to profiles and subordinate Tecs as well (Month 2)	…whether there should be icons to link Tecs to a user's profile (Month 2)	…under which circumstances a Tec title is still available for a user (Month 1)	…under which circumstances the use count of a Tec increases (Month 2)
…how the Tec with the lowest page number should be ordered (Month 2)	…the rules that describe under which circumstances which Tecs or profiles should be seen (Month 4)	..which color a Tec should have when the Tec already exists in the same column (Month 3)	…whether the new design will have icons for users to click on (Month 3)	…how a user can select an existing weblink (Month 4)	…how to count Tecs that are selected by a user (Month 2)
…how switching between pages of unselected Tecs pages affect their order in the columns (Month 5)	…under which circumstances which background should be shown (Month 4)	…which color Tecs should have when they are linked to a user's profile (Month 3)	…how the new layout for private profiles differs from the old one, i.e. what changes need to be made (Month 5)	…why the term changed from "group profile" to "Tec groups" (Month 4)	…the reason for why the Tec count should increase (Month 2)

Appendix 2: Detailed Example for Designing Domain (Color)

A software developer initially faced a boundary related to color in the 3rd month of the project when he was not clear about which color a Tec should have when the Tec already exists in the same column. While the developer visually pointed to the example in the WP, the client used the WP to explain what color the Tec should have in this situation. Another semantic boundary that was similar to the previous one then occurred in the same month when the team member worked on another requirement. For example, seven days after the prior episode, a developer was not clear about which color a Tec should have when it is linked to a user's profile. The developer opened the current prototype (WP) to demonstrate a scenario of a Tec that is linked to a user's profile. The client then explained using the WP that Tecs linked to a user profile are green and added a similar comment to the user story. Yet, a similar semantic boundaries – also related to colors – re-occurred several months later. For example, a team member did not understand under which circumstances connecting lines have certain colors. The developer re-used the screen shots (BP) from an earlier episode to formulate questions about the groups and private profiles (e.g., "*Why two lines here?")*. The developer pointed to the screen shot to ask about the behaviour of connecting lines.Then the developer opened the current prototype (WP) displaying what he has implemented so far and contrasted it with the not yet implemented requirement represented by the screen shot (BP). The client explained that the private profile is the only element that is shown by all three colored lines since *"it is always visible to everybody, but at the same time it belongs to the user."* Further similar semantic boundaries (i.e., color) re-occurred in the same month.

References

Awad, M. (2005). *A comparison between agile and traditional software development methodologies.* School of Computer Science and Software Engineering. Crawley, Perth, University of Western Australia.

Baba, M. L., J. Gluesing, H. Ratner, and K. H. Wagner (2004). "The contexts of knowing: Natural history of a globally distributed team." *Journal of Organizational Behavior* 25 (5), 547-587.

Barrett, M. and E. Oborn (2010). "Boundary object use in cross-cultural software development teams." *Human Relations* 63 (8), 1199-1221.

Bechky, B. A. (2003). "Sharing meaning across occupational communities: The transformation of understanding on a production floor." *Organization Science* 14 (3), 312-330.

Berczuk, S. (2007). "Back to basics: The role of agile principles in success with a distributed scrum team." In: *Proceedings of the Agile Conference*. Washington: DC, pp. 382-388.

Bergman, M., J. L. King and K. Lyytinen (2002). "Large-scale requirements analysis revisited: The need for understanding the political ecology of requirements engineering." *Requirements Engineering* 7 (3), 152-171.

Boland, R. J., Jr. and R. V. Tenkasi (1995). "Perspective Making and Perspective Taking in Communities of Knowing." *Organization Science* 6 (4), 350-372.

Cao, L. and B. Ramesh (2007). "Agile software development: Ad hoc practices or sound principles?" *IT Professional* 9 (2), 41-47.

Carlile, P. R. (2002). "A pragmatic view of knowledge and boundaries: Boundary objects in new product development." *Organization Science* 13 (4), 442-455.

Carlile, P. R. (2004). "Transferring, translating, and transforming: An integrative framework for managing knowledge across boundaries." *Organization Science* 15 (5), 555-568.

Carmel, E. and P. Tjia (2005). *Offshoring information technology: sourcing and outsourcing to a global workforce.* Cambridge, UK: Cambridge University Press.

Corbin, J. and A. Strauss (1990). *Basics of qualitative research: Grounded theory procedures and techniques.* Thousand Oaks, CA: Sage Publications.

Cramton, C. D. (2001). "The mutual knowledge problem and its consequences for dispersed collaboration." *Organization Science* 12 (3), 346-371.

Damian, D. E. and D. Zowghi (2003). "RE challenges in multi-site software development organisations." *Requirements Engineering* 8 (3), 149-160.

Danait, A. (2005). "Agile offshore techniques - a case study", In: *Proceedings of the Proceedings of the Agile Development Conference.* Denver: CO, pp. 214-217.

Dibbern, J., J. Winkler and A. Heinzl (2008). "Explaining variations in client extra costs between software projects offshored to India." *MIS Quarterly* 32 (2), 333-366.

Dingsøyr, T., S. Nerur, V. Balijepally and N. B. Moe (2012). "A decade of agile methodologies: Towards explaining agile software development." *Journal of Systems and Software* 85 (6), 1213-1221.

Dougherty, D. (1992). "Interpretive barriers to successful product innovation in large firms." *Organization Science* 3 (2), 179-202.

Dyba, T. and T. Dingsoyr (2009). "What do we know about agile software development?" *Software, IEEE* 26 (5), 6-9.

Faraj, S. and Y. Xiao (2006). "Coordination in fast-response organizations." *Management Science* 52 (8), 1155-1169.

Floyd, C. (1984). *A systematic look at prototyping. Approaches to prototyping*. Ed. by R. Budde, K. Kuhlenkamp, L. Mathiassen and H. Züllighoven. Berlin: Springer, pp. 1-18.

Fowler, M. and J. Highsmith (2001). "Agile methodologists agree on something." *Software Development* 9 (8), 28-32.

Gorschek, T. and C. Wohlin (2006). "Requirements abstraction model." *Requirements Engineering* 11 (1), 79-101.

Henderson, K. (1991). "Flexible sketches and inflexible data bases: Visual communication, conscription devices, and boundary objects in design engineering." *Science, Technology & Human Values* 16 (4), 448-473.

Herbsleb, J. D. and D. Moitra (2001). "Global software development." Software, IEEE 18 (2), 16-20.

Highsmith, J. (2013). *Adaptive software development: a collaborative approach to managing complex systems.* Boston, MA: Addison-Wesley.

Hull, E., K. Jackson and J. Dick (2011). *Requirements Engineering.* 3rd Ed. London: Springer Science & Business Media.

Jarvenpaa, S. L. and D. E. Leidner (1998). "Communication and trust in global virtual teams." *Journal of Computer‐Mediated Communication* 3 (4), 791-815.

Kautz, K., K. Kuhlenkamp, and H. Züllighoven (1992). *Prototyping–An Approach to Evolutionary System Development.* Berlin: Springer.

Kellogg, K. C., W. J. Orlikowski and J. Yates (2006). "Life in the trading zone: Structuring coordination across boundaries in postbureaucratic organizations." *Organization Science* 17 (1), 22-44.

Kliem, R. (2004). "Managing the risks of offhore IT development projects." *Information Systems Management* 21 (3), 22-27.

Kotlarsky, J., H. Scarbrough and I. Oshri (2014). "Coordinating expertise across knowledge boundaries in offshore-outsourcing projects: The role of codification." *MIS Quarterly* (forthcoming).

Kruchten, P. (2001). "Agility with the RUP." *Cutter IT Journal* 14 (12), 27-33.

Langley, A. (1999). "Strategies for theorizing from process data." *Academy of Management Review* 24 (4), 691-710.

Levina, N. and E. Vaast (2005). "The emergence of boundary spanning competence in practice: implications for implementation and use of information systems." *MIS Quarterly* 29 (2), 335-363.

Levina, N. and E. Vaast (2008). "Innovating or doing as told? Status differences and overlapping boundaries in offshore collaboration." *MIS Quarterly* 32 (2), 307-332.

Lyytinen, K. and M. Newman (2008). "Explaining information systems change: a punctuated socio-technical change model." *European Journal of Information Systems* 17 (6), 589-613.

Majchrzak, A., A. Malhotra, and R. John (2005). "Perceived individual collaboration know-how development through information technology–enabled contextualization: evidence from distributed teams." *Information Systems Research* 16 (1), 9-27.

Majchrzak, A., P. H. More and S. Faraj (2012). "Transcending knowledge differences in cross-functional teams." *Organization Science* 23 (4), 951-970.

Meso, P. and R. Jain (2006). "Agile software development: adaptive systems principles and best practices." *Information Systems Management* 23 (3), 19-30.

Naur, P. (1984). *Comments on "On the Psychology of Prototyping" by Anker Helms Jörgensen. Approaches to Prototyping.* Ed. by R. Budde, K. Kuhlenkamp, L. Mathiassen and H. Züllighoven. Springer, pp. 290-291.

Nerur, S. and V. Balijepally (2007). "Theoretical reflections on agile development methodologies." *Communications of the ACM* 50 (3), 79-83.

Nicolini, D., J. Mengis and J. Swan (2012). "Understanding the role of objects in cross-disciplinary collaboration." *Organization Science* 23 (3), 612-629.

Opie, C. and P. J. Sikes (2004). *Doing educational research*. London, UK: SAGE Publications.

Orlikowski, W. J. (2002). "Knowing in practice: Enacting a collective capability in distributed organizing." *Organization Science* 13 (3), 249-273.

Rowen, R. B. (1990). "Software project management under incomplete and ambiguous specifications." *IEEE Transactions on Engineering Management* 37 (1), 10-21.

Sarker, S. and S. Sahay (2004). "Implications of space and time for distributed work: an interpretive study of US–Norwegian systems development teams." *European Journal of Information Systems* 13 (1), 3-20.

Sarker, S., S. Sarker, A. Sahaym and N. Bjørn-Andersen (2012). "Exploring value cocreation in relationships between an ERP vendor and its partners: A revelatory Case study." *MIS Quarterly* 36 (1).

Schmickl, C. and A. Kieser (2008). "How much do specialists have to learn from each other when they jointly develop radical product innovations?" *Research Policy* 37 (6), 1148-1163.

Schwaber, K. and M. Beedle (2002). *Agile software development with Scrum.* Upper Saddle River, NJ: Prentice Hall.

Seidel, V. P. and S. O'Mahony (2014). "Managing the Repertoire: Stories, Metaphors, Prototypes, and Concept Coherence in Product Innovation." *Organization Science* 25 (3), 691-712.

Shen, Z., K. Lyytinen and Y. Yoo (2014). "Time and information technology in teams: a review of empirical research and future research." *European Journal of Information Systems*, 1-27.

Sørensen, E. (2009). *The materiality of learning: Technology and knowledge in educational practice.* New York City, NY: Cambridge University Press.

Star, S. L. (2010). "This is not a boundary object: Reflections on the origin of a concept." *Science, Technology & Human Values* 35 (5), 601-617.

Star, S. L. and J. R. Griesemer (1989). "Institutional ecology, translations' and boundary objects: Amateurs and professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39." *Social Studies of Science* 19 (3), 387-420.

Tiwana, A. (2004). "Beyond the black box: knowledge overlaps in software outsourcing." *IEEE Software* 21 (5), 51-58.

Trompette, P. and D. Vinck (2009). "Revisiting the notion of Boundary Object." *Revue d'anthropologie des connaissances* 3 (1), 3-25.

Vlaar, P. W., P. C. van Fenema and V. Tiwari (2008). "Cocreating understanding and value in distributed work: How members of onsite and offshore vendor teams give, make, demand, and break sense." *MIS Quarterly* 32 (2), 227-255.

von Krogh, G., C. Rossi-Lamastra and S. Haefliger (2012). "Phenomenon-based research in management and organisation science: When is it rigorous and does it matter?" *Long Range Planning* 45 (4), 277-298.

Winkler, M., T. Huber and J. Dibbern (2014). "The Software Prototype as Digital Boundary Object – A Revelatory Longitudinal Innovation Case." *Thirty Fifth International Conference on Information Systems (ICIS)*. Auckland: NZ.

Yakura, E. K. (2002). "Charting time: Timelines as temporal boundary objects." *Academy of Management Journal* 45 (5), 956-970.

Yin, R. K. (2009). *Case study research: Design and methods. Thousand Oaks*. CA: Sage Publications.