The Software Prototype as Digital Boundary Object – A Revelatory Longitudinal Innovation Case

Completed Research Paper

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

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

Jens Dibbern
University of Bern
jens.dibbern@iwi.unibe.ch

Abstract

With the availability of lower cost but highly skilled software development labor from offshore regions, entrepreneurs from developed countries who do not have software development experience can utilize this workforce to develop innovative software products. In order to succeed in offshored innovation projects, the often extreme knowledge boundaries between the onsite entrepreneur and the offshore software development team have to be overcome. Prior research has proposed that boundary objects are critical for bridging such boundaries – if they are appropriately used. Our longitudinal, revelatory case study of a software innovation project is one of the first to explore the role of the software prototype as a digital boundary object. Our study empirically unpacks five use practices that transform the software prototype into a boundary object such that knowledge boundaries are bridged. Our findings provide new theoretical insights for literature on software innovation and boundary objects, and have implications for practice.

Keywords: Software innovation, boundary object, distributed team

Introduction

Suppose there is an entrepreneur in a western country with a visionary idea for a digital innovation. The entrepreneur is rather a novice in the development of digital technology and financial resources are limited. Ten years back, realizing such a vision may have remained a dream, as software development workforce was expensive and getting the commitment of others to join such a project difficult given the vagueness of the vision. Today, however, a skilled software development workforce is available in low wage countries such as India, China, the Philippines or Vietnam at significantly lower cost than domestically (Gefen and Carmel 2008). If our entrepreneur now hires an offshore IT vendor, how can this vision become a reality given that successful innovation projects require the entrepreneur and the software developers to bridge knowledge boundaries?

Prior research has suggested that for bridging knowledge boundaries the use of objects such as sketches and design drawings is essential since objects help to convey and translate ideas between team members with heterogeneous knowledge stocks (e.g. Barley et al. 2012; Bechky 2003b; Bowker and Star 1999; Henderson 1991; Star and Griesemer 1989). This stream of research has emphasized the importance of the way in which objects are being used. For instance, some studies found that the same object might help bridging a boundary at one time, while strengthening boundaries at another time – depending on how the object was used (e.g. Barrett and Oborn 2010; Barrett et al. 2012). However, it is not yet understood why using objects in different ways has different consequences for bridging knowledge boundaries. In this study we argue that this can be better understood if we revisit two basic tenets of boundary object theory. First, boundary object theory suggests conceiving objects as multi-dimensional (Star 2010). In offshored software development, different types of prototypes such as paper prototypes, mock-up designs, and partly functioning software prototypes have been identified as key objects mediating interactions between the onshore and the offshore team (Srikanth and Puranam 2011). Thus, prior research has focused on multiple different objects (e.g. Carlile 2002; Henderson 1991). We develop a multi-dimensional conceptualization of software prototypes that integrates those objects under one theoretical umbrella. Second, boundary object theory holds that an object only helps bridging knowledge boundaries if it qualifies as a *boundary* object, i.e. if it develops the distinct object properties of being "plastic enough to adapt to local needs [...] yet robust enough to maintain a common identity across sites" (Star and Griesemer 1989, p. 393). While prior research on object use in software development has borrowed arguments from boundary object theory to make sense of their results (e.g. Bechky 2003b; Bergman et al. 2007; Carlile 2002; Kellogg et al. 2006; Levina and Vaast 2005), to the best of our knowledge no study has investigated whether or not objects facilitating the bridging of boundaries actually exhibit the theoretically predicted object properties.

Therefore, it is the goal of this study to better understand how software prototypes are used, how such use practices affect the emergence of those properties that transform a simple object into a boundary object, and whether or not such differences in prototype use and properties explain differences in bridging knowledge boundaries. To achieve this goal we conducted a longitudinal, revelatory case study of a software innovation project between a Swiss entrepreneur and a Vietnamese vendor company. Based on rich observational data, extensive document analysis and complementary in-depth interviews we explore how differences in how the software prototype is used lead to different object properties with different consequences for bridging boundaries. The findings of our study contribute to research on IS innovation, IS outsourcing, boundary objects and practice. The next section discusses the theoretical background of this study and establishes a research framework that guides our empirical analysis. Subsequently, the study's research design is described. Finally, our results are presented and discussed.

Related Literature & Theoretical Background

Knowledge Boundaries in Software Innovation Projects

Most innovation happens at the boundaries between specialized pools of knowledge (Carlile 2004; Leonard-Barton 1995). Yet, specialized pools of knowledge necessarily imply knowledge boundaries between the holders of heterogeneous knowledge stocks. Three knowledge boundaries are distinguished (Carlile 2002): First, the syntactic knowledge boundary refers to informational differences between actors – they do not share a common syntax or language. This is a common problem in software innovation projects where developers are necessarily unfamiliar with the innovation to be developed and typically

collaborate with the entrepreneur for the first time (Souder and Moenaert 1992). For instance, in order to express the novel ideas incorporated in the innovative software product, the entrepreneur might have developed a new vocabulary that is difficult to understand for the offshored software development team. Second, the semantic boundary is concerned with interpretive differences among team members – they do not share a common understanding and, therefore, can interpret things differently. In software innovation this phenomenon manifests as inconsistent interpretations of desired product qualities. For instance, team members from different cultures may attach different meanings to product and project attributes causing misunderstandings about desired functionality and the project progress (Levina and Vaast 2008). Third, the pragmatic boundary refers to different goals among team members who do not share common interests (Carlile 2002). For example, the entrepreneur might want the team to engage in risky exploratory search for novel solutions while the development team prefers to implement requirements with the least effort possible. While specialized pools of knowledge are essential for software innovation, past research has also emphasized that the potential benefits of such knowledge heterogeneity can only be realized if knowledge experts engage in activities to bridge them (Guinan et al. 1998).

Bridging Knowledge Boundaries – The Role of Using (Boundary) Objects

Traditionally, research on bridging knowledge boundaries has emphasized the need for deep, longwinded and effortful dialogue in order to confront and eventually reduce knowledge differences (Boland and Tenkasi 1995; Hargadon and Bechky 2006; Majchrzak et al. 2012; Tsoukas 2009). More recently, it has been proposed that knowledge boundaries can be more effectively bridged if in this process, specialists actively use objects (e.g. Kellogg et al. 2006; Levina and Vaast 2005; Majchrzak et al. 2012). In this stream of research a wide array of objects has been studied including accounting ledgers, standardized reporting forms, documents, drawings, protocols, repositories, and organizational models (Bechky 2003b) used to visualize and textually represent ideas (Majchrzak et al. 2012). In the IS literature examples of objects that help bridge knowledge boundaries include document archives, enterprise resource planning systems, prototypes, proto architectures, and project plans (e.g. Bergman et al. 2007; Briers and Chua 2001; Levina and Vaast 2005; Park and Boland ; Pawlowski and Robey 2004). This stream of research was mainly concerned with the consequences of different object use practices. For instance, prior research has revealed that team members can present boundary objects in different ways such that they either help to create a shared understanding or enforce own interests (Barley et al. 2012; Seidel and O'Mahony 2014). Likewise, prior studies have found that some ways of using objects are more conducive to mediate relations between experts (Bechky 2003b; Majchrzak et al. 2012) while other use practices would be geared to strengthen the boundaries between groups (Barrett and Oborn 2010; Bechky 2003a; Levina and Vaast 2005).

While research on object use has unveiled and established the importance of objects for cross-boundary collaboration, the findings of this research stream suffer from fragmentation and potential contradictions. In particular, it seems virtually impossible to compare the findings of prior research because a huge variety of different objects was studied without systematically capturing the properties that enable an object to become a boundary object. That is to say, prior research was less "concerned with the essence of objects themselves" (Ewenstein and Whyte 2009, p. 3). Yet, object properties are at the core of boundary object theory that has inspired most of the above mentioned work. In their seminal article Star and Griesemer (1989, p. 393; p. 408) [emphasis added] state the properties of those objects that help bridging knowledge boundaries, and therefore qualify as boundary objects as follows:

> "Boundary objects are objects which are both *plastic* enough to adapt to local needs and the constraints of the several parties employing them, yet *robust* enough to maintain a common identity across sites [...] their boundary nature is reflected by the fact that they are simultaneously concrete and abstract [...]"

The distinction between plastic and robust refers to the capacity of the object to adapt to local needs (Star and Griesemer 1989, p. 393). If adaptable the object is plastic; if unchangeable, it is robust. The distinction between abstract and concrete refers to the degree to which an object is represented through symbolic or through material representations. If the object is represented by symbolic elements it is abstract; if it is represented through material representations it is concrete (cf. Table 1 for definitions and example codes). In line with boundary object theory, we believe that considering these object properties could help to explain why different objects have different consequences for bridging knowledge boundaries. However,

the boundary object concept has not only inspired many researchers but also caused some confusion culminating in the question: "how can [boundary objects] be simultaneously concrete and abstract? The same and yet different?" (Star 2010, p. 614). Star (2010, p. 763) herself points to the seemingly paradoxical nature of the boundary concept when admitting that her descriptions of the concept are "stuck with using Newtonian language for quantum phenomena." However, she also advises a solution for this problem when noting that the concept is "n-dimensional" (Star 2010 p.603) in nature. Next, we adopt this idea by introducing a multi-dimensional conceptualization of the focal object of this study – the software prototype.

An Integrative Conceptualization of Software Prototypes

A software prototype is an incomplete version of a software system (Sommerville 2004, p. 409). During the software development process, software systems move "through a series of representations from requirements to finished code" (Ramesh et al. 2012, p. 323). This series of representations relates to different variants of prototypes that resemble the complete software system with different degrees of fidelity (Pohl 2007). For instance, early on in the development process a system's architecture may be represented through a UML diagram, the initial concept of a user interface through a paper prototype, and the graphical appearance through a mock-up design. Later on in the development process, when parts of the system are already coded, this partially working system becomes the prototype with some already working functionality.

Informed by boundary object theory, prior IS research has already studied different prototype variants as separate objects (e.g. D'Adderio 2001; Im et al. 2003). Since it is our goal to integrate fragmented research we follow the advice of Star and more general methodological recommendations for studying paradox (Lewis 2000) by combining different objects in one integrative multi-dimensional conceptualization. In particular, we conceptualize the prototype as consisting of two analytically separate but co-evolving sub dimensions (Lyytinen and Newman 2008). On the one hand, the *working prototype* is that part of the prototype that has already been developed and implemented, and is thus represented through the material software prototype (adapted from Lyytinen and Newman 2008). On the other hand, the *building prototype* is the more hypothetical part of the prototype that needs to be erected to carry out and plan the software development task (adapted from Lyytinen and Newman 2008). It is represented through written software requirements, use case diagrams, design drawings and mock-up designs (cf. Table 1 for definitions and example codes).

	Key Concept	**Definition**	**Example Indicators**	**Coding Example**
Knowledge Boundaries	Syntactic boundary (based on Carlile 2002)	Developers and entrepreneur do not share a common language, i.e. they are not familiar with the language used	Different terminologies used; (mutually) unintelligible accents used; unfamiliarity with the language used by the other side	The developer uses the term "brick" whereas the entrepreneur uses the word "Neo" to refer to the same feature; developers are not familiar with the term "subordinate" in an user story description
	Semantic boundary (based on Carlile 2002)	Developers and entrepreneur do not share a common understanding, i.e. they interpret the same things differently	One side of the boundary lacks understanding of the purpose of a functionality / the behavior of a functionality; both sides of the boundary are at cross-purposes	The developer asks: "What can users do with Neo groups? What is the function of the Neo groups? We don't understand..."; the developer thinks that the color of a particular Neo should be green, whereas the entrepreneur thinks it should be grey

	Pragmatic boundary (based on Carlile 2002)	Developers and entrepreneur do not share common interests, i.e. a clash of interests occurs	Conflicting views on role responsibilities; reluctance to compromise by disregarding what the other side pursues	The entrepreneur asks: "When is the right moment to change to the final design? You are the developer, you need to know". The developer says "you tell me, it is up to you" to decide; the developer wants the entrepreneur to test a new functionality but the entrepreneur does not want to
Dimensions of software prototype	Building prototype (based on Lyytinen et al. 2008)	The building prototype refers to the not yet implemented hypothetical dimension of the prototype that is embodied through symbolic represen-tations of software requirements	Symbolic representations of software requirements that express what and how they should be implemented in the future such as user stories, acceptance criteria, design drawings, mock-up designs, use case diagrams, etc.	A developer opens a user story description; the entrepreneur opens a mock-up design; the developer opens a use case diagram
	Working prototype (based on Lyytinen et al. 2008)	The working prototype refers to the already implemented dimension of the prototype that is embodied through a functioning piece of software	Functioning piece of software that can be used and tested in a dedicated test environment (e.g. test server); Completely implemented functions / methods / classes / designs.	The developer opens the already implemented functionality of Neo groups; the developer opens the already implemented new design for making payments
Object properties	Plasticity (based on Star 2010; Star et al. 1989)	Plasticity refers to the degree to which an object has the capacity to adapt to the specific local needs of one or both sides of the knowledge boundary. High levels of plasticity are referred to as "plastic", low levels as "robust"	*Increase in plasticity ("Making plastic"):* Questions, feedback and suggestions for change invited; uncertainty about functionality is expressed; future changes of a functionality are announced; developers are afforded to change requirements to facilitate implementation *Decrease in plasticity ("Making robust"):* An underlying logic is emphasized and presented as unchangeable by pointing to misalignment; Developers are not afforded to change requirements to facilitate implementation	*Increase in plasticity ("Making plastic")*: The entrepreneur asks the developers about an already implemented function: "What do you think about the function? Should we change it?" *Decrease in plasticity ("Making robust"):* When the developer shows an already implemented functionality, the entrepreneur points out that it is wrong that the user sees the content of Z in the right column

	Abstraction (based on Star 2010; Star et al. 1989)	Abstraction refers to the degree to which an object is represented through codified and symbolic representations rather than through concrete and material representations. High levels of abstraction are referred to as "abstract", low levels as "concrete"	*Increase abstraction ("Making abstract"):* Referring to unknown (not yet specified/not yet discussed) requirements *Decrease in abstraction ("Making concrete"): A requirement* is specified further by adding additional information to its description / by attaching screenshots / by making interdependencies with another requirement explicit	*Increase in abstraction ("Making abstract"):* The entrepreneur explains that in the future something will change without outlying how it will change *Decrease in abstraction ("Making concrete"):* The entrepreneur adds additional information to a user story description

Table 1. Key Concepts of the Study

Research Framework

In order to elucidate how and why software prototypes help bridging knowledge boundaries in software innovation projects, we integrate the above-mentioned theoretical elements – i.e. object properties and object dimensions – into a research framework (Figure 1). This will allow us to capture subtle differences between object dimensions at one point in time, and changes of objects over time: First, differences between the two prototype dimensions will be captured as differences in object properties (plastic/robust, abstract/concrete). In particular, the building prototype consists of symbolic representations of the to-be-developed software system that can be rapidly created and adapted. The working prototype, by contrast, consists of already implemented software code that is more effortful to adapt and create but that is also already a more material representation of the desired end product. Thus, building prototypes tend to be more abstract and plastic than the relatively concrete and robust working prototypes. Second, we will describe changes of the prototype as changes in object properties. For instance, as the innovation project progresses, existing requirements are implemented, and parts of the building prototype are transformed into parts of the working prototype. As a consequence, the working prototype may become more concrete and robust over time. Likewise, as existing requirements of the building prototype are refined and further specified over the course of the project, the building prototype becomes more concrete and robust.

At the same time, however, the ability to function as a boundary object hinges not only on externally given object properties but on how an object is used in the daily practice of software development teams (e.g. Barrett and Oborn 2010; Levina and Vaast 2005; Majchrzak et al. 2012; Nicolini et al. 2012; Seidel and O'Mahony 2014). Such differences in prototype use will be captured in two ways. First, we will describe differences in prototype use by capturing which prototype (working prototype, building prototype) is used by team members, i.e. whether the building prototype, the working prototype or both are used to bridge boundaries. However, since prior research has found the same objects to have highly different consequences for bridging knowledge boundaries, depending on how the object was used (Levina and Vaast 2005), it seems necessary to more deeply explore this "how" by carving out differences in empirically observed object use practices. Also, if objects with similar properties do indeed have different consequences for bridging knowledge boundaries, it seems likely that the way how objects are used affects the perception of their properties. For instance, a working prototype with a perfectly working set of functionalities might be perceived as less robust if the team members frame these functionalities as still changeable. Therefore, we will interpret whether object use practices change the perception of object properties.

In sum, we strive for a better understanding of how and why the use of the prototype helps bridging different knowledge boundaries (syntactic, semantic, pragmatic). The "how" part of this question will be answered by exploring the use practices of different objects (building prototype / working prototype). The "why" part of this question will be answered by drilling down to the "essence" of boundary objects, i.e. by assessing the properties (and their changes) that qualify an object as *boundary* object. This will answer questions like: Does bridging knowledge boundaries indeed require objects with distinct properties? Does

bridging different boundaries require different object properties? Or more generally, why are some object use practices more or less effective than others?

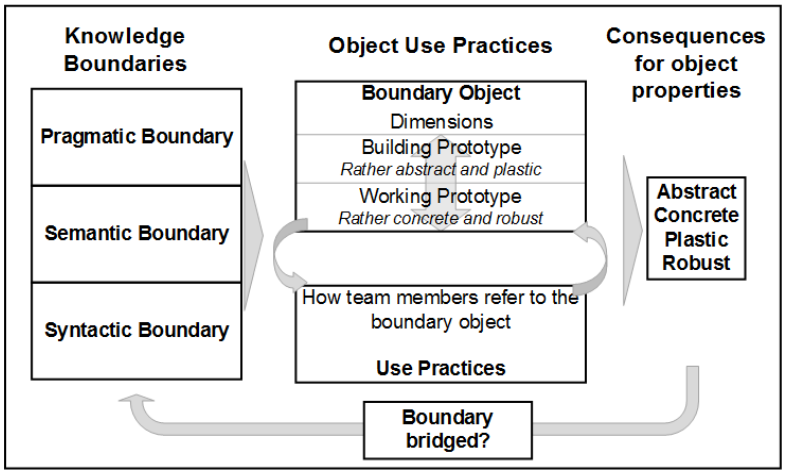

Figure 1. Research Framework

Research Methods

Case Selection, Data Collection and Empirical Context

Recognizing the paucity of in-depth field studies on using software prototypes as boundary objects, our strategy was to study a thus far unexplored case in-depth (Sarker et al. 2012; Yin 2009). To ensure appropriateness of such a single-case research design, a case that would be potentially unique, exemplary and revelatory was purposefully selected (Patton 2002; Yin 2009). A Vietnam-based software development company Vietnamsoft (pseudonym) gave us the opportunity to select such a project from their portfolio. The small software company has extensive experience in offshore software development and has already conducted more than a 100 projects with customers in Switzerland. From Vietnamsoft's project portfolio we chose the Neoproduct case because it suited our research objectives exceptionally well: A Swiss architect without any software development background decided to quit his job and to become a software entrepreneur. For this purpose he hired Vietnamsoft (6 team members). His business idea which we refer to as "Neoproduct" is to be a web-based application. Neoproduct shall – once finished - enable users to store and share a large variety of different information and media with each other in a new way (e.g. links, videos, photos, etc.). In particular, users should be able to access compressed information contained by a "Neo." A "Neo" is an information carrier that can be created by users. Each Neo has three standardized fields (title, description, photo) plus additional fields depending on the type of information stored (e.g. a Video-Neo has a field for YouTube URLs).

While the entrepreneur had thought about his new idea for several years (when he was still an architect), the developers are totally unfamiliar with the idea. Accordingly, the project members compared Neoproduct with software products as different as a "Facebook imitation", a "new form of Wikipedia", or a "new kind of internet". Yet, in contrast to the entrepreneur, Vietnamsoft's employees possess the specific development expertise needed to implement web-based applications (e.g. HTML 5.0, Java, JavaScript, PHP, database technology, etc.). Against this backdrop, the knowledge boundaries between the entrepreneur and the development team that this study is interested in are rather extreme, thus, promising *unique* insights. Moreover, at Neoproduct, novelty arises throughout the innovation project as the entrepreneur further develops his idea. This is typical for innovation projects (Langlois 2002; Schmickl 2006), making it an *exemplary* case for this domain (Yin 2009). Also, the Neoproduct case is potentially revelatory because few researchers have investigated object use practices (i.e. using the prototype with its different dimensions and properties) and the consequences for bridging knowledge boundaries with such a rich data set.

In particular, we received access to three longitudinal data sources: First, we gathered rich observational data of all (virtual) meetings that took place between the Swiss-based and the Vietnam-based team members over a period of 6 months. The first and the second author of this paper attended 50 virtual meetings as non-participant observers, recorded them, and took field notes. This resulted in a total of 19,6 hours of observation and 204 pages of observation protocols. Our second data source is the log files and the content exchanged via a task and code-management tool called Assembla. In Assembla, all software requirements such as descriptions and use case diagrams are represented through certain attributes such as a status, a priority and possibly some discussion surrounding the requirement. Changes in attributes and representations over the course of the project were tracked and entered into a database. Our third data source is supplementary retrospective interviews with all project participants that took place every couple of months. In sum, 12 interviews have been conducted. These three longitudinal data sources not only provided us with exceptionally rich insights, they also ensured "variance" with respect to the key concepts of our study despite only investigating a single-case. In particular, they allowed us to systematically compare different object use practices and their consequences over time (instead of across cases) (Yin 2009). Finally, the small development team has adopted the agile software development methodology Scrum with rapid prototyping at its core (Schwaber and Beedle 2002) rendering the Neoproduct case *exemplary* (Yin 2009) for extensively relying on an object (i.e. the software prototype) in cross-boundary collaboration. This cross-boundary collaboration occurs through virtual meetings between the entrepreneur and the development team. During those virtual meetings the entrepreneur and the development team can talk to each other (using Skype) and share their screen (using WebEx). Screen sharing is essential for cross-boundary collaboration because it allows the entrepreneur and the development team to make the same software prototype visible to everyone. In particular, the working prototype being discussed is made visible to all by screen sharing, a browser window that runs Neoproduct. The building prototype is made visible by sharing the Assembla application where all requirements such as use cases, design drawings, etc. are stored.

Data Analysis

We followed an iterative three-stage process of data analysis and theory building where findings of earlier stages inform later stages and vice versa (Huber et al. 2013; Miles and Huberman 1994).

Stage 1 - Identifying Episodes

In a first step, all observation protocols were carefully reviewed to identify episodes (Lyytinen and Newman 2008). The definition of an episode follows our initial theoretical considerations (cf. Figure 1, Table 1): An episode is an ordered sequence of events (Langley 1999; Lyytinen and Newman 2008) where the project members used the prototype to bridge a knowledge boundary. Thus, an episode always begins with a knowledge boundary (syntactic, semantic, pragmatic) between the entrepreneur and the software developers. Then, to bridge the knowledge boundary, team members engage in different object use practices with the software prototype. We applied a stopping rule to elucidate the end of an episode: An episode ends when the two sides of the boundaries stop using the prototype, or shift attention to another boundary. At the end of each episode we assessed whether and to what extent the previously identified boundary was better understood than at the beginning of the episode.

Stage 2 - Analyzing Object Use Practices and their Consequences

The goal of this stage was to map each episode into the elements of our research framework. First, for each episode the observed or reported behaviors of how the entrepreneur and developers used the prototype were coded. This coding was, on the one hand, informed by our research framework (Yin 2009), i.e. for each activity it was analyzed as to whether the rather robust and concrete working prototype, the more plastic and abstract building prototype, or both (i.e. prototype dimensions) were used. On the other hand, our coding of use behaviors remained open for novel insights emerging from our data (Corbin and Strauss 1990). This allowed us to characterize use behaviors not only in terms of which prototype dimensions were used but also how they were used. Such empirically observed use behaviors were openly coded and then grouped together to similar concepts that we call activities (cp. Table 2 for an overview of these activities). Then, we assessed the consequences of such object use behaviors in two steps. First, we checked whether the prototype itself was adapted and whether such adaptations led to a change in object properties (e.g. whether the prototype had become more robust and/or more concrete). This step also involved a careful examination of perceptual changes in object properties occurring through distinct use behaviors (e.g. if

openness for adaptation of an already implemented feature was emphasized). Second, consequences for bridging knowledge boundaries were analyzed: Verbal statements during meetings or in the collaboration tool that indicated increased understanding were coded as bridging a knowledge boundary while verbal statements indicating the opposite were coded as failure to bridging a boundary. Table 1 provides additional details of how data was coded during this stage. To ensure robustness of our analysis we made heavy use of data triangulation. Results of this analysis stage were captured in a first case write-up and tabular data displays (Miles and Huberman 1994).

Stage 3 - Comparing and Theorizing

During this stage all episodes were systematically compared with regards to the elements of our research framework (boundary object properties/dimensions, use practices, knowledge boundaries) to identify recurring patterns across episodes (Yin 2009). The goal was to identify object use practices that were most conducive for bridging knowledge boundaries. This was achieved in two steps during which we made extensive use of data displays and tables (Miles and Huberman 1994). First, we compared use activities and sought for differences in their effectiveness for bridging knowledge boundaries. Once such differences were identified, we sought for similarities between use activities geared towards bridging the same knowledge boundary, and for differences between use activities that were geared towards bridging different boundaries. In this second step, we analyzed similarities and differences in terms of what prototype dimensions were used, how they were used and which consequences this use had for (perceived) changes in object properties. As a result of this iterative process, use activities were abstracted into five higher-level categories with explanatory power (Corbin and Strauss 1990) that we call object use practices. These five object use practices are the main findings of our study and are presented next.

Findings

We identified five use practices that were applied by members of the Neoproduct case to bridge different knowledge boundaries. Next, these five practices will be presented. For the sake of clarity, we have grouped the use practices according to the particular knowledge boundary they help to bridge. First, we will provide a general description of the knowledge boundary. Second, we will present each use practice by describing their main activities and will ground each practice in our data by illustrating it with a real-world example from the Neoproduct case.

Use Practices for Bridging Syntactic and Semantic Knowledge Boundaries

These use practices were mostly applied when the entrepreneur and the development team faced differences with regards to language (syntactic boundary) or had different interpretations or understandings about features, functionalities, or visual designs (semantic boundary).

> Syntactic Boundary: *"[Often] the [entrepreneur] uses another wording and term."* [Main Developer, Interview 1]
>
> Semantic Boundary: "*There are misalignments [...] at the beginning I did not understand [the requirement] correctly."* [Main Developer, Interview 2]

Whenever problems with regard to language or understanding occurred, the use practices (1) *Contrasting the building prototype with the working prototype*, (2) *Visually pinpointing and exemplifying in the building or working prototype* and (3) *Verbally relating a concept in the working or building prototype to the whole* helped to bridge them. Each use practice is presented next.

Use Practice 1: Contrasting the Building Prototype with the Working Prototype

The use practice was particularly helpful when software developers attempted to understand how information is shared between different users to clarify complex connections between users and Neos: "*Especially in the Neo Project, the idea is quite complex. I mean the features, what matches what is not easy to understand for some cases - like how they share a Neo from one user to another user - in public mode, in group mode [...] it is complex. So the challenge is to make sure the team, which is far away from him [the entrepreneur], not face to face...has an understanding.*" This use practice involves several activities where the team members contrast the building with the working prototype, i.e. team members switch between more abstract elements such as screen shots, tables, and user story descriptions and

contrast them with more concrete elements such as an already implemented functionality, feature or visual design. This switching between abstract and concrete elements helped bridging syntactic and semantic boundaries because the concrete implementation seemed to reduce some of the complexity arising from the severe knowledge asymmetry between the project participants:

> *"It is always a combination [of objects that help to] become clearer. I sense that it is essentially more complex than expected at first sight [...]."* [Entrepreneur, Interview 2]

In particular, the already implemented working prototype served as a reference point allowing the team members to exactly point out which aspects of the thus far unclear requirement of the working prototype were similar or even identical to the concrete implementation, and what aspects were different. This subsequently allowed the team members to further specify the not yet implemented requirement. Hence, through this practice the rather abstract building prototype is made more concrete. Next we illustrate Use Practice 1 with an example from the Neoproduct case.

Knowledge boundary: One user story required the developers to never display the same Neo to the same user at the same time. However, the developer struggled to implement this requirement since he did not understand what information should be displayed instead once the duplicated Neo would be deleted (semantic boundary).

Use Activities: The developer opened a rough diagram (building prototype) displaying different connections between Neos and user profiles (Figure 2). He used this diagram to present, in a first step, which Neo would be displayed twice given the relations specified in the diagram. Then, in order to find out which Neo should be displayed instead once such duplicated Neos are not displayed anymore, the developer opened the working prototype such that both, the diagram and the working prototype, were shown simultaneously on one screen. The developer used the concrete functionality of the working prototype to display a number of duplicated Neos, and to elucidate which other Neos (that are also linked to the given user) should be displayed once the duplications are removed. This aided the discussion because the developer was able to understand the "rule" stipulating the correct surrogate-Neo.

Consequences: The knowledge that was obtained was then used by the developer to further specify the not yet implemented diagram (building prototype) by adding supplementary descriptions to the diagram (cf. Figure 2 to see the diagram before and after the specification), i.e. the diagram is made more concrete. The semantic boundary was bridged since the developer then understood that the information of the duplicated Neo is only shown once after its deletion.

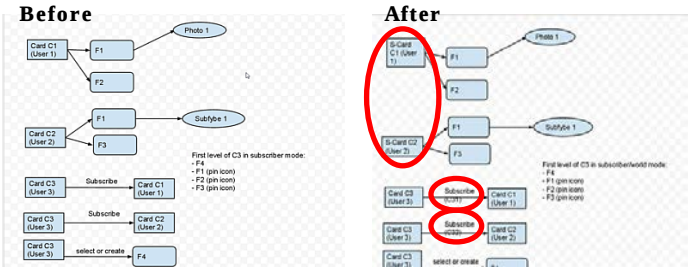

Figure 2. Diagram before and after specification

Use Practice 2: Visually Pinpointing and Exemplifying in the Building or Working Prototype

This use practice was particularly helpful when developers were lacking understanding of functionality, features and, most importantly, of visual designs [semantic boundary]. The prototype helped closing such gaps.

> *„It is simply a visual medium, so it is then clear, it is the fastest way. [...] I could also write an A4-page text but then I would have to write it down. [Instead] when you click at that point and on top to the left etc. it is faster"* [Developer, Interview 2]

This use practice encompasses several activities such as visually locating and pointing to fragments in rather abstract not yet implemented requirements (screen shots, user story descriptions, etc.) or in concrete already implemented functionalities, features or visual designs. In order to visually locate and point out gaps in understanding, developers then move their mouse cursor to the relevant part of either the building or the working prototype, they visually highlight a text in the user story description, or an icon in a visual design (e.g. by drawing circles around the icon). Subsequently, the obtained information is then used to further specify the building prototype, i.e. the building prototype is made more concrete. In other instances, the developers additionally go through example cases in the already functioning software (working prototype) by clicking on the current functionalities to understand certain behaviors of Neos or user profiles depending on certain conditions (e.g. User 1 is linked to Neo 3). Thus, they further specify the already implemented functionality. Next we illustrate Use Practice 2 with an example from the Neoproduct case.

Knowledge boundary: One user story required the developer to improve the layout for Neos. However, the developer struggled to implement this requirement since he did not understand what it looks like when a Neo is "shadowed" (semantic boundary).

Use activities: The developer opened the screen shots (building prototype) attached to the requirement displaying the new layout of Neos including a visualization of a "shadowed" Neo. Then, the developer uses these abstract visual designs represented through the screen shots to point the mouse cursor to a "shadowed" Neo. The entrepreneur explained: in the visual example in the right column the Neo is not linked to the user profile; in the left column, then the shadow will move behind the user's profile. The developer visually located these visual examples of "shadowed" Neos while the entrepreneur described them to further specify the requirement.

Consequences: The knowledge that was obtained was subsequently used by the developer to further specify the not yet implemented layout (building prototype) by adding additional comments to the requirement. The semantic boundary is bridged since the developer then understood that a "shadow" is behind a Neo or a user profile.

Use Practice 3: Verbally Relating a Concept in the Working or Building Prototype to the Whole

This use practice was applied by the entrepreneur when the developers faced more general understanding problems (semantic boundary), i.e. problems that were not limited to one requirement but rather concerned with the more abstract "big picture" or "visions behind ideas" [Entrepreneur, Interview 1]. When faced with such semantic boundaries the entrepreneur engaged in various prototype-related activities. One activity involves vaguely announcing future changes of an already implemented functionality by referring to concepts of the future which are yet unknown or not yet specified (i.e. abstract) for the developers. Yet, most of the times the above described semantic boundaries were bridged when the entrepreneur created new, rather abstract representations (building prototype) of the big picture or the vision (i.e. tables, figures, drawings, descriptions). The Scrum Master mentions that using such abstract representations was very helpful for bridging semantic boundaries, which is confirmed by the entrepreneur:

> *"It was probably a total of three tables and their combination that has resulted in an understanding for developers."* [Entrepreneur, Interview 2]

Thus, to bridge semantic boundaries that are concerned with abstract problems, abstract representations of the building prototype seem particularly helpful. However, while helpful for bridging semantic boundaries concerned with more abstract ideas, software developers often faced difficulties to apply these ideas to individual requirements. In these cases, use practice 3 was usually followed by use practice 1 and 2 striving for an exact specification of the consequences of an abstract idea for an individual requirement. Next we illustrate Use Practice 3 with an example from the Neoproduct case.

Knowledge boundary: A user story required the developer to implement a functionality that calculates the intensity of colors for Neos and user profiles. A wide range of other requirements were affected by this rather abstract, general idea. The developer struggled to implement the requirement since he did not understand under which conditions the intensity of colors for Neos or profiles should change (semantic boundary).

Use activities: The developer opened the table attached to the user story (building prototype) prepared by the entrepreneur. The table displayed different columns with descriptions and visual designs needed to be able to calculate the correct color intensity for Neos and profiles. Based on the table, the entrepreneur explained the abstract idea and the reason behind it: "*When browsing on Neoproduct.com there will be brighter or darker areas*". The user should be able to recognize which Neos or user profiles are visited by many users: "*because behind every Neo there are people associated - represented by the profile count.*" Thus, there will be "*famous Neos and other ones which a user uses alone.*" [Entrepreneur, statement during meeting] Thus, when the user scrolls to the top of the page, the brighter the color of Neos will become as these are visited by many users whereas the Neos at the bottom are dark (i.e. less frequently visited by users).

Consequences: The knowledge that was obtained through this object use was then utilized by the developer to consider the more abstract, "big picture" of the idea. In a next step, the entrepreneur further specified how the developers can read the abstract table (i.e. use practice 2). The semantic boundary was bridged since the developer then understood that the intensity of colors reveals if a Neo or a user profile is visited often or not.

Use Practices for Bridging Pragmatic Knowledge Boundaries

These use practices were primarily applied to circumvent conflicts of interests (pragmatic boundary). Most of the time these conflicts of interest were concerned with the entrepreneur's primary objective – i.e. to make his idea real as fast as possible without having to invest more than already covered through the fixed-price contract underlying the partnership with Vietnamsoft. This, however, would sometimes be difficult to achieve if requirements of the building prototype were to remain unchanged:

> "[...] *sometimes requirements are very difficult to implement. [The entrepreneur] would have to pay more and it would take more time. We cannot do everything we want. Actually, there are limitations of the technology as well.*" [Main Developer, Interview 1]

In other cases, it may happen that the entrepreneur expects the developer to engage in certain activities as part of his role whereas the developer expects the entrepreneur to take care of these activities. For example, the developer wants the entrepreneur to make the decision about the expected deadline for the implementation of the final design, and vice versa. Likewise, the two sides of the boundary may also disagree on how user stories should be implemented and how much efforts are needed to implement a user story. For instance, while the developer tries to keep the efforts low and facilitate the implementation process, the entrepreneur is more interested in innovative solutions. When faced with such goal conflicts the use practices (4) *verbally emphasizing openness for adaptation in the building or working prototype* and (5) *verbally narrowing down the scope for changes in the building or working prototype* helped to bridge the pragmatic boundary. These use practices will be presented next.

Use practice 4: Verbally Emphasizing Openness for Adaptation in the Building or Working Prototype

This use practice encompasses a number of activities. For the entrepreneur these activities are usually related to the building prototype. For instance, the entrepreneur may display a given requirement (building prototype) that, from his point of view, is already completely formulated but invites the development team to ask critical questions, or to suggest alternatives that would be easier to implement:

> "*Here you have to help me what's the best way to develop it. I am not a developer [...] I have no idea of what is technically possible.*" [Entrepreneur, Interview 2]

> "*What do you think? Is it ok to display all possible navigation items?*" [Entrepreneur, Interview 2]

Another activity to emphasize openness for adaptation is to postpone implementation but explicitly record this "as a limitation for now". This involves that the entrepreneur explicitly verifies the limitation by creating a follow-up user story assigning him with the task to think about how the requirement may be changed. Also, developers emphasize openness for adaptation. In other situations, when the entrepreneur wants a certain solution to be implemented for a requirement (building prototype), but the developers have concerns about the technical feasibility, they suggest implementing an alternative solution – and the entrepreneur usually accepts such adaptations:

> *In this case, "[the entrepreneur] is very open for change [...] so we adapt another solution that is easier to implement."* [Main Developer, Interview 2]

In many instances, developers also want the working prototype to appear open for adaptation. In particular, the developers often make a rather robust already implemented functionality, feature or visual design to appear open for change. This activity is usually applied to deal with the verification vs. validation trade-off: While developers are usually quite confident that their implementation complies with the requirements (verification), they often experience uncertainty as to whether their implementation meets the actual customer needs (validation). Therefore, developers constantly emphasize the provisional character of the working prototype by asking the entrepreneur for any changes he desires. Activities include, for example, inviting questions or feedback on already implemented functions:

> *"in the prototype we have the opportunity to show what we are doing. So to check if it is correct what we are doing for the requirement."* [Main developer, Interview 2]

Irrespective of whether team members refer these activities to the working or the building prototype, emphasizing openness for adaptation seemed to make the prototype appear more plastic. Next we illustrate Use Practice 4 with an example from the Neoproduct case.

Knowledge boundary: A requirement stipulated the developers to implement a feature that would show each user of Neoproduct.com a personalized stream of activities. The entrepreneur has formulated this requirement as one large user story, yet, the developer is interested in splitting it into sub-requirements since "if the user story is smaller it is very easy to test" [Main Developer, Interview 1] (pragmatic boundary).

Use activities: The developer opened the user story (building prototype) displaying the user story description to express the need to split the requirement into two sub-user stories that would be easier to test independently (i.e. one user story for visual design, the other for implementation of features) Thus, he asked the entrepreneur for changes in the requirement to facilitate its implementation. Looking at the written requirements of the building prototype, the entrepreneur not only allowed these changes but even offered to carry out this task himself.

Consequences: The written requirement of the working prototype is made more plastic by verbally expressing openness for and by encouraging additional future changes. The pragmatic boundary was bridged since the entrepreneur was open for a compromise and then splitted the requirement himself.

Use Practice 5: Verbally Narrowing Down the Scope for Changes in the Building or Working Prototype

This use practices was primarily applied by the entrepreneur: "*[...] It is his vision, I think the man with the vision knows how the vision should be done."* [Designer, Interview 1] In a similar vein, the entrepreneur envisions himself as the "*thinker [...] in this project*" [Entrepreneur, Interview 1], and applies this practice when he believes that he knows "*how to go about this project*" [Entrepreneur, Interview 1]. This use practice encompasses several activities that are all geared towards narrowing down the scope for changes in the building or working prototype. One of those activities involves displaying the working prototype and verbally emphasizing that the current implementation is wrong since it is misaligned with an unchangeable logic: "*We are not allowed to see visibility icons for third party card [...] I can never adjust visibility settings of someone else's cards.*" [Entrepreneur, statement during meeting] Thus, the entrepreneur does not allow the developers to make changes to the requirement and instead enforces his own interests:

> *"I think you have to trust me now. This will work. Even if you don't understand it right now- in 4 weeks you will."* [Entrepreneur, Interview 1]

Accordingly, when the entrepreneur verbally narrows down the scope for changes in the building or working prototype, the rather plastic, not yet implemented requirement (building prototype) or a relatively robust, already implemented functionality (working prototype) becomes more robust. Making the prototype more robust by narrowing down the scope for changes helped the entrepreneur to sustain his interests in terms of desired functionality. Next we illustrate Use Practice 5 with an example from the Neoproduct case.

Knowledge boundary: After developers started to implement new functionalities and visual designs, they expected the entrepreneur to test these functionalities on the test server after the meeting to receive feedback from the entrepreneur on their work in "progress". However, the entrepreneur was more interested in testing already completed implementations (pragmatic boundary).

Use activities: The developer opened his "work in progress" in the current prototype (working prototype) displaying what was implemented so far of the requirement. He roughly presented the current state of the functionality and visual design to the entrepreneur. However, the entrepreneur immediately started emphasizing that the layout of this "work in progress" is misaligned with an important logic from the building prototype. In particular, the coloration of Neos did not follow a previously specified logic. The entrepreneur emphasized the importance of this logic that would stipulate coloration not only for this requirement but for many more in the future, i.e. he emphasized the robustness of the logic.

Consequences: Emphasizing the robustness of the logic by pointing to the current misalignment of the already implemented design to the unchangeable logic helped the entrepreneur to enforce his own interests because these observed misalignments with the logic provided him with a good reason to decide that in would indeed be too early to test the functionality in the test server. The pragmatic boundary was bridged since the entrepreneur then decided that he will not test the functionality at this point in time.

Table 2 provides an overview of all five use practices, most important activities, as well as their consequences for object properties and bridging knowledge boundaries.

Use practices	**Main activities**		**Assessment of change of property in prototype dimension**	**Assessment of effect on bridging Knowledge boundary**
	Entrepreneur	Developers		
Use practice 1: Contrasting the *building prototype* with the *working prototype*	-Further specifying a not yet implemented requirement (building prototype) by contrasting it with an already implemented feature, function or visual design (working prototype)	-Further specifying a not yet implemented requirement (building prototype) by contrasting it with an already implemented feature, function or visual design (working prototype)	The rather abstract building prototype is made more concrete	These use practices help bridging syntactic and semantic boundaries.
Use practice 2: Visually pinpointing and exemplifying in the *building or working prototype*	-Going through visual examples and scenarios to explain/answer questions about the behavior of a functionality or visual design -Visually locating specific position of parts of a design which have or have not yet been implemented -Directing the developers to the relevant parts of a not yet implemented or already implemented visualization (e.g. right column) or text (e.g. user story descriptions, tables)	-Going through examples and scenarios in the building prototype or working prototype (e.g. click on functions) to understand the behavior of a functionality or visual design -Visually highlighting the requirements/a written question/screen shots of a particular user story -Visually locating and pointing to fragments of designs or functionalities which have or have not yet been implemented	The abstract building prototype or the concrete working prototype is made more concrete	

Use practice 3: Verbally relating a concept in the *working or building prototype* to the whole	-Referring to concepts of the future which are yet unknown / not specified for the developers -Referring to unknown user stories/tables/concepts -Explaining the general logic of a concept or idea -Explaining the reason/meaning behind a concept or idea		Mixed: Abstract building or concrete working prototype might become more or less concrete depending on how abstract knowledge is used subsequently	This use practice helps bridging semantic boundaries when combined with use practice 1 or 2.
Use practice 4: Verbally emphasizing openness for adaptation in the *building or working prototype*	-Expressing uncertainty whether the described functionality / design / etc. is reasonable -Announcing that the described functionality / design etc. will change -Suggesting / allowing changes and/or undertaking changes in requirements to facilitate implementation or improve the user experience/software quality -Inviting feedback, discussions and inputs on not yet implemented functionalities and also already implemented	-Asking for /suggesting changes in requirements -Changing requirements to facilitate implementation -Expressing uncertainty whether a requirement will work -Inviting questions and feedback on already implemented functions	The abstract building prototype or concrete working prototype is made more plastic	This use practice helps bridging pragmatic boundaries
Use practice 5: Verbally narrowing down the scope for changes in the *building or working prototype*	-Emphasizing that the current implementation/wording or not yet implemented screenshot is wrong since it is misaligned with an unchangeable logic from the building prototype -Not allowing developers to change requirements to facilitate an implementation -Rejecting suggestions or indicating disapproval of proposed/asked by others		The abstract building prototype or concrete working prototype is made more robust	This use practice helps bridging pragmatic boundaries

Table 2. Object Use Practices Transforming Software Prototypes into Boundary Objects

Discussion

It is the goal of this study to better understand which object use practices transform prototypes into boundary objects such that knowledge boundaries are bridged. Therefore, we longitudinally investigated a single case guided by a research framework that integrated seminal conceptual work on boundary objects with empirical research on object use. At the core of this research framework is an integrative conceptualization of software prototypes outlining two dimensions and their properties. Therefore, this study is the first to not only investigate the use of different objects, but also "the essence" of the used

objects. Our case study unveiled five object use practices. These five practices have in common that they are all related to the software prototype and that they all help bridging boundaries occurring over the course of the offshored innovation project. Yet, the five practices also exhibit interesting differences: While Practice 1, Practice 2 and Practice 3 were more conducive for bridging semantic and syntactic boundaries; Practice 4 and practice 5 were more conducive for bridging pragmatic boundaries. Interestingly, the capacity of object use practices for bridging different knowledge boundaries also systematically varied with regards to object properties. Practices geared towards bridging semantic and syntactic boundaries were more concerned with the abstract/concrete property of the software prototype (Practice 1/2/3), while practices geared towards bridging pragmatic boundaries were more concerned with the plastic/robust-property (Practice 4/5). Thus, overall, our findings suggest that an object can help bridging a variety of different knowledge boundaries because an object can exhibit different properties depending on how it is used. In particular, the two sides of the boundaries can engage in use practices that refer to, emphasize and combine different object dimensions with different object properties such that, depending on the use practice those properties that qualify an object as a boundary object in a given situation are highlighted or fade to the background. These findings bear important implications for research on software innovation projects, IS outsourcing, boundary objects and for practice. These are discussed next.

Discovering Effective Object Use Practices

Prior conceptual research on boundary objects has argued that objects with different capacities are required depending on the type of boundary faced (Carlile 2004, p. 565; Swan et al. 2007). However, prior empirical research on object use did not explicitly address this claim because different object use practices were not systematically linked to bridging different types of knowledge boundaries (e.g. Levina and Vaast 2005; Majchrzak et al. 2012; Nicolini et al. 2012; Seidel and O'Mahony 2014). Thus, unveiling object use practices and linking them to different types of knowledge boundaries, as it has been done in this study, represents a theoretical contribution in its own right. In addition, the findings obtained through our study refine research on object use in two important ways. First, prior research has argued that different objects are required to bridge different knowledge boundaries (Carlile 2002). Our findings have shown that the software prototype had the capacity to help bridging all three types of boundaries. This suggests that it is possible to unify all the capacities needed to bridge all three boundaries in one single object, instead of different objects with different capacities. Our study also offers a novel explanation for this capacity of boundary objects: The properties that qualify a software prototype as a boundary object are not materially fixed properties of the object itself but they also emerge through use practices. In particular, our findings unveil that the multi-dimensional nature of software prototypes allowed team members to highlight those dimensions needed to bridge a boundary, and suppress others that were not needed or that might even be detrimental for a given boundary. As an example, in many instances, team members from different sides of the knowledge boundary were able to arrive at a sufficient common understanding about desired product qualities – thus bridging semantic boundaries – by combining the concrete working prototype with the abstract building prototype. In contrast, other use practices were more conducive to bridge other boundaries. For example, verbally framing the building prototype as open for adaptation (i.e. making the building prototype increasingly plastic) was particularly helpful to circumvent pragmatic boundaries.

Our findings also contribute to the stream of research that has already acknowledged that use practices of artifacts are key for their capacity to function as boundary objects (e.g. Levina and Vaast 2005; Majchrzak et al. 2012; Nicolini et al. 2012; Seidel and O'Mahony 2014). However, these studies have not explicitly captured object properties – a concept that is at the core of boundary object theory. In this respect, our empirical exploration of use practices *and* object properties goes beyond and thereby complements prior object use studies. In addition, most literature has conceptualized effective boundary objects as concrete (Bechky 2003b; Carlile 2002; Henderson 1991). While our findings do not refute such claims, they have also shown that the abstract/concrete dimension is only important for bridging semantic and syntactic boundaries. In contrast, the plastic/robust dimension that has mostly been neglected seems important for bridging pragmatic boundaries.

Exploring the Co-Existence of Seemingly Paradoxical Boundary Object Properties

Our results reveal that bridging different boundaries require different object properties. As mentioned in our theory section, Star (2010, p. 763) has pointed to the seemingly paradoxical nature of the boundary

concept and suggested that they are multidimensional and "based in action" (Star 2010, p. 603). However, previous research did not explicitly investigate the "under theorized and underexplored" question of whether and how boundary objects can indeed exhibit contradictory object properties – as suggested by boundary object theory. Our study contributes to this question in three important ways. First, boundary objects can simultaneously be abstract and concrete, or robust and plastic because the object itself consists of different dimensions - with one dimension being abstract (e.g. the building prototype) and the other being concrete (e.g. the working prototype), or one dimension being plastic and the other robust. This complements Ewenstein and Whyte (2009, p. 8) who have also conceptualized objects as multidimensional and argued that through this multidimensionality objects can "bridge between the concrete and the abstract." While our findings do not refute this claim, they also show that whether and to what extent objects exhibit these polarized properties depends on how they are used: Object dimensions with polarized properties can be combined through use practices such that the whole prototype is indeed both, abstract and concrete (cp. Contrasting the building prototype with the working prototype). Second, actors can manipulate the only seemingly fixed properties of any object dimension through verbal strategies. For instance, something seemingly robust, like a requirement of the building prototype, that has already been agreed upon can be framed as plastic by verbally inviting changes (cp. verbally emphasizing openness for adaptation in the building or working prototype). Thus, boundary objects can have polarized properties because objects that exhibit one property pole (e.g. robust) can be "flavored" with the opposite pole (e.g. plastic) through verbal strategies. Third, one boundary object can have polarized properties since one object exhibits those properties at different points in time. This is an insight acquired through the longitudinal character of our study: By splitting our case into temporally ordered episodes (cp. Data Analysis section) we were able to observe such dynamics (cf. Table 2). In particular, the same boundary object (the prototype) exhibited different properties in each episode – depending on how it was used. Thus, prototype properties oscillate between episodes where one property is highlighted (e.g. making robust), and phases where the opposite property is highlighted (e.g. making plastic).

Practical Implications

Recent IS outsourcing research has already emphasized the need to effectively manage offshore vendors over the course of a project to inhibit project failures (Gregory et al. 2013; Huber et al. 2013). Since knowledge boundaries are one reason for such project failures, our findings provide meaningful guidance for practitioners on how to bridge these boundaries through meaningful use of the software prototype. Since object use practices have consequences on the objects properties (i.e. abstract, concrete, plastic, robust) needed for bridging different knowledge boundaries, practitioners are advised to use prototypes consciously – depending on the boundary they are facing. The use practices that transform software prototypes into boundary objects that have been identified by this study can be purposefully enacted by professionals who deal with problems in language, understanding and interest across boundaries. For example, the developers or Scrum Master can interfere in meetings and make sure that they engage in appropriate object use practices. If they follow this advice, then, projects may come to success without the need to engage in extensive knowledge sharing - even if the involved individuals possess highly heterogeneous knowledge.

Limitations and Future Research

One limitation of this study is that we have focused on one single case. While we have argued that the longitudinal character and the uniqueness of the case provided us with the opportunity to acquire novel theoretical insights, future studies should determine whether the practices we unveiled are also present in other contexts and under different conditions. Furthermore, cultural, educational and status differences may have influenced our results (Dibbern et al. 2008; Levina and Vaast 2008). Moreover, our study focused on those use practices that were actually conducive for bridging boundaries. While we believe that the novel theoretical insights provided through this study justify this focus, future studies should take a more holistic approach and systematically compare practices that are successful with practices that fail in bridging knowledge boundaries. Furthermore, future research should try to capture potential co-evolutionary dynamics between the prototype and use practices. In particular, as the innovation project progresses the properties of the software prototype may also change – possibly affecting its capacity to act as a boundary object.

References

Barley, W.C., Leonardi, P.M., and Bailey, D.E. 2012. "Engineering Objects for Collaboration: Strategies of Ambiguity and Clarity at Knowledge Boundaries," *Human Communication Research* (38:3), pp. 280-308.

Barrett, M., and Oborn, E. 2010. "Boundary Object Use in Cross-Cultural Software Development Teams," *Human Relations* (63:8), pp. 1199-1221.

Barrett, M., Oborn, E., Orlikowski, W.J., and Yates, J. 2012. "Reconfiguring Boundary Relations: Robotic Innovations in Pharmacy Work," *Organization Science* (23:5), pp. 1448-1466.

Bechky, B.A. 2003a. "Object Lessons: Workplace Artifacts as Representations of Occupational Jurisdiction," *American Journal of Sociology* (109:3), pp. 720-752.

Bechky, B.A. 2003b. "Sharing Meaning across Occupational Communities: The Transformation of Understanding on a Production Floor," *Organization Science* (14:3), pp. 312-330.

Bergman, M., Lyytinen, K., and Mark, G. 2007. "Boundary Objects in Design: An Ecological View of Design Artifacts," *Journal of the Association for Information Systems* (8:11), pp. 547-568.

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

Bowker, G.C., and Star, S.L. 1999. *Sorting Things Out: Classification and Its Consequences*. Cambridge, MA: MIT Press.

Briers, M., and Chua, W.F. 2001. "The Role of Actor-Networks and Boundary Objects in Management Accounting Change: A Field Study of an Implementation of Activity-Based Costing," *Accounting, Organizations and Society* (26:3), pp. 237-269.

Carlile, P.R. 2002. "A Pragmatic View of Knowledge and Boundaries: Boundary Objects in New Product Development," *Organization Science* (13:4), pp. 442-455.

Carlile, P.R. 2004. "Transferring, Translating, and Transforming: An Integrative Framework for Managing Knowledge across Boundaries," *Organization Science* (15:5), pp. 555-568.

Corbin, J., and Strauss, A. 1990. *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. Thousand Oaks, CA, US: Sage Publications.

D'Adderio, L. 2001. "Crafting the Virtual Prototype: How Firms Integrate Knowledge and Capabilities across Organisational Boundaries," *Research Policy* (30:9), pp. 1409-1424.

Dibbern, J., Winkler, J., and Heinzl, A. 2008. "Explaining Variations in Client Extra Costs between Software Projects Offshored to India," *MIS Quarterly* (32:2), pp. 333-366.

Ewenstein, B., and Whyte, J. 2009. "Knowledge Practices in Design: The Role of Visual Representations Asepistemic Objects'," *Organization Studies* (30:1), pp. 7-30.

Gefen, D., and Carmel, E. 2008. "Is the World Really Flat? A Look at Offshoring at an Online Programming Marketplace," *MIS Quarterly* (32:2), pp. 367-384.

Gregory, R.W., Beck, R., and Keil, M. 2013. "Control Balancing in Information Systems Development Offshoring Projects," *MIS Quarterly* (37:4), pp. 1211-1232.

Guinan, P.J., Cooprider, J.G., and Faraj, S. 1998. "Enabling Software Development Team Performance During Requirements Definition: A Behavioral Versus Technical Approach," *Information Systems Research* (9:2), pp. 101-125.

Hargadon, A.B., and Bechky, B.A. 2006. "When Collections of Creatives Become Creative Collectives: A Field Study of Problem Solving at Work," *Organization Science* (17:4), pp. 484-500.

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), pp. 448-473.

Huber, T., Fischer, T., Dibbern, J., and Hirschheim, R. 2013. "A Process Model of Complementarity and Substitution of Contractual and Relational Governance in Is Outsourcing," *Journal of Management Information Systems* (30:3), pp. 81-114.

Im, S., Nakata, C., Park, H., and Ha, Y.-W. 2003. "Determinants of Korean and Japanese New Product Performance: An Interrelational and Process View," *Journal of International Marketing* (11:4), pp. 81-112.

Kellogg, K.C., Orlikowski, W.J., and Yates, J. 2006. "Life in the Trading Zone: Structuring Coordination across Boundaries in Postbureaucratic Organizations," *Organization Science* (17:1), pp. 22-44.

Langley, A. 1999. "Strategies for Theorizing from Process Data," *Academy of Management Review* (24:4), pp. 691-710.

Langlois, R.N. 2002. "Modularity in Technology and Organization," *Journal of Economic Behavior & Organization* (49:1), pp. 19-37.
Leonard-Barton, D. 1995. *Wellsprings of Knowledge: Building and Sustaining the Sources of Innovation*. Boston, MA: Harvard Business Press.
Levina, N., and Vaast, E. 2005. "The Emergence of Boundary Spanning Competence in Practice: Implications for Implementation and Use of Information Systems," *MIS Quarterly* (29:2), pp. 335-363.
Levina, N., and Vaast, E. 2008. "Innovating or Doing as Told? Status Differences and Overlapping Boundaries in Offshore Collaboration," *MIS Quarterly* (32:2), pp. 307-332.
Lewis, M.W. 2000. "Exploring Paradox: Toward a More Comprehensive Guide," *Academy of Management Review* (25:4), pp. 760-776.
Lyytinen, K., and Newman, M. 2008. "Explaining Information Systems Change: A Punctuated Socio-Technical Change Model," *European Journal of Information Systems* (17:6), pp. 589-613.
Majchrzak, A., More, P.H., and Faraj, S. 2012. "Transcending Knowledge Differences in Cross-Functional Teams," *Organization Science* (23:4), pp. 951-970.
Miles, M.B., and Huberman, A.M. 1994. *Qualitative Data Analysis: An Expanded Sourcebook*. Thousand Oaks, CA: Sage Publications.
Nicolini, D., Mengis, J., and Swan, J. 2012. "Understanding the Role of Objects in Cross-Disciplinary Collaboration," *Organization Science* (23:3), pp. 612-629.
Park, J., and Boland, R. "Boundary Objects as Action in Information Systems Development (ISD): A Dramaturgical Perspective Using Sociodrama," *AMCIS 2012 Proceedings*.
Patton, M.Q. 2002. "Two Decades of Developments in Qualitative Inquiry a Personal, Experiential Perspective," *Qualitative Social Work* (1:3), pp. 261-283.
Pawlowski, S.D., and Robey, D. 2004. "Bridging User Organizations: Knowledge Brokering and the Work of Information Technology Professionals," *MIS Quarterly* (28:4), pp. 645-672.
Pohl, K. 2007. *Requirements Engineering: Grundlagen, Prinzipien, Techniken*. Heidelberg: dpunkt. verlag.
Ramesh, B., Mohan, K., and Cao, L. 2012. "Ambidexterity in Agile Distributed Development: An Empirical Investigation," *Information Systems Research* (23:2), pp. 323-339.
Sarker, S., Sarker, S., Sahaym, A., and Bjørn-Andersen, N. 2012. "Exploring Value Cocreation in Relationships between an ERP Vendor and Its Partners: A Revelatory Case Study," *MIS Quarterly* (36:1).
Schmickl, C. 2006. *Organisationales Lernen in Innovationssystemen: Eine Empirische Analyse der Entstehung und Umsetzung von Wissen*. Marburg: Tectum-Verlag.
Schwaber, K., and Beedle, M. 2002. *Agile Software Development with Scrum*. Upper Saddle River, NJ: Prentice Hall.
Seidel, V.P., and O'Mahony, S. 2014. "Managing the Repertoire: Stories, Metaphors, Prototypes, and Concept Coherence in Product Innovation," *Organization Science* (25:3), pp. 691-712.
Sommerville, I. 2004. *Software Engineering. International Computer Science Series*. Boston, MA: Addison Wesley.
Souder, W.E., and Moenaert, R.K. 1992. "Integrating Marketing and R&D Project Personnel within Innovation Projects: An Information Uncertainty Model," *Journal of Management Studies* (29:4), pp. 485-512.
Srikanth, K., and Puranam, P. 2011. "Integrating Distributed Work: Comparing Task Design, Communication, and Tacit Coordination Mechanisms," *Strategic Management Journal* (32:8), pp. 849-875.
Star, S.L. 2010. "This Is Not a Boundary Object: Reflections on the Origin of a Concept," *Science, Technology & Human Values* (35:5), pp. 601-617.
Star, S.L., and Griesemer, J.R. 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), pp. 387-420.
Swan, J., Bresnen, M., Newell, S., and Robertson, M. 2007. "The Object of Knowledge: The Role of Objects in Biomedical Innovation," *Human Relations* (60:12), pp. 1809-1837.
Tsoukas, H. 2009. "A Dialogical Approach to the Creation of New Knowledge in Organizations," *Organization Science* (20:6), pp. 941-957.
Yin, R.K. 2009. *Case Study Research: Design and Methods*. Thousand Oaks, CA: Sage Publications.