December 2003

Knowledge Partitioning in Outsourced Software Development: A Field Study

Amrit Tiwana
Emory University

KNOWLEDGE PARTITIONING IN OUTSOURCED SOFTWARE DEVELOPMENT: A FIELD STUDY

Amrit Tiwana
Emory University
Atlanta, GA USA
atiwana@bus.emory.edu

Abstract

The outsourced software development process has traditionally relied on a requirements-driven black-box approach for transferring knowledge of customer needs to vendors. When this approach is feasible, the need for the customer and the vendor to deeply understand each others' knowledge domain is limited. We describe this as symmetric *division of knowledge. However,* asymmetric *overlaps in knowledge are necessary at the vendor-customer boundary in projects involving conceptual or process newness.*

In this study, we examine the conditions under which overlaps in knowledge at the vendor-customer boundary are necessary for enhancing the development process in outsourcing relationships. We develop and test a model using data collected in a large-scale field study of 209 software projects in 209 software development organizations belonging to three of the largest global software consortia.

The study makes three contributions: (1) we empirically demonstrate that it is more important for a vendor to possess a higher level of business knowledge in conceptually new projects and for the customer to have a higher level of technical knowledge when the project involves process newness, (2) we assess the effectiveness of various integrating mechanisms, and (3) we show that there are potential downsides to blindly increasing vendor-customer overlaps in knowledge beyond those that have traditionally characterized software development.

Introduction

Software development is a knowledge intensive process that involves the coordinated application of a variety of specialized knowledge in devising a coherent software solution for a business problem. Two types of knowledge are germane to this process: (1) knowledge about the application problem domain and (2) technical knowledge through which a software solution is developed. Traditionally, this process has relied on formal requirements for transferring knowledge about the design problem from the customer to the software development organization (hereinafter vendor). If the formal requirements can accurately capture the needs of the customer, the benefits of the vendor having in-depth knowledge about the business domain or the customer about the technical intricacies of the software (e.g., programming languages, development methodologies, and the software development process) are limited. Both the vendor and the customer can then enjoy the benefits of specializing in their activities and need little knowledge about each others' activities for successful software development. However, under some conditions, such absence of overlap between the customer's and developer's knowledge can lead to poor software design that fails to embody the customer's needs.

When is it beneficial for a customer to have deep technical knowledge and the developer to have deep knowledge about the business application domain? When do vendor-customer overlaps in knowledge about each others' activities help ensure that the

resulting software accurately embodies the customer's needs? This is an understudied issue that can potentially provide insights into the design of successful outsourcing relationships.

In this paper, we explore this issue through a large-scale field study of 209 software projects in 209 software development organizations belonging to three of the largest global software consortia. We draw on knowledge management theory to develop a theoretically grounded model that not only explains how such knowledge partitioning influences software development but also *why* it does so. In addition to examining empirically the effects of the distribution patterns of business and technical knowledge at the vendor-customer boundary, we also examine the effectiveness of various mechanisms that can facilitate integration of such knowledge during the software development process.

The contributions of this paper are three-fold. First, it empirically demonstrates the conditions under which vendor-customer knowledge overlaps facilitate and impede knowledge integration. We find that it is more important for a vendor to possess a higher level of business application domain knowledge in conceptually new projects and for the customer to have a higher level of technical knowledge when the project involves process newness. This evidence for the contingencies under which asymmetrical overlaps in knowledge improve the software development process is a unique contribution to the literature. Second, we empirically assess the *relative* effectiveness of development coordination tools, customer-vendor interactions, architectural design effort, use of mature software processes, and prior experience in the software development process. Third, we show that there are potential downsides to blindly increasing overlaps in knowledge beyond those that have traditionally characterized software development. Given the global nature of our data, the insights drawn from this study are safely generalizable to software development in a variety of contexts.

The rest of the paper is organized as follows. In the next section, we review the prior literature. We then develop the research model, and describe the research methodology and data collection. We then describe the analyses and results. We conclude with the implications of our findings for software development research and practice.

Prior Literature

Knowledge in Software Development

The role of knowledge in software development is widely recognized (Adelson and Soloway 1985; DeSouza 2003; Robillard 1999; Rus and Lindvall 2002; Walz et al. 1993). The software development process can be conceptualized as a knowledge intensive activity of organizing and integrating the specialized expertise, skills, and perspectives of various project stakeholders into an appropriate, coherent, and practical solution (Constantine and Lockwood 1993; Faraj and Sproull 2000; Rus and Lindvall 2002). Rus and Lindvall (2002) identify two types of knowledge that software development draws on: (1) technical knowledge that is used to develop a system and (2) knowledge about the business application domain of a system. Technical knowledge refers to knowledge about design (e.g., design patterns, heuristics, best practices, technical constraints, and estimation models), programming (e.g., programming languages and development tools), and software processes (e.g., methodology, code testing and debugging procedures). The development of a system is based on such knowledge. Business application domain knowledge refers to knowledge about the customer's business processes, business rules, activities, stakeholder needs, and the customer business objectives for the software. Decomposition of the design problem, acceptable design tradeoffs, and interdependencies with other systems in the customer organization are based on the latter type of knowledge.

Knowledge Integration in Software Development

Having access to technical expertise and application domain knowledge is necessary, but the two must be *coherently integrated and embodied in the design of software* for it to meet the customer's needs (Faraj and Sproull 2000). Knowledge about customer needs must be embodied in the conceptual design, functionality, and features of the system as well as in intermediate design artifacts such as contracts, project plans, requirements, and software specifications (Armour 2000; Cantor 2002; Robillard 1999; Walz et al. 1993). We define this process of embodying business application domain knowledge with technical knowledge in the design of the software as *knowledge integration*. Such integration of business application domain knowledge with technical knowledge is the key mechanism through such knowledge is applied to develop an appropriate software design (Ramesh 2002; Ramesh and Dhar 1992; Walz et al. 1993). Both empirical and case studies have shown that such integration of knowledge enhances both software development effectiveness and efficiency (Curtis et al. 1988; Faraj and Sproull 2000; Walz et al. 1993).

However, knowledge integration continues to be a chronic challenge for software development (DeSouza 2003; Ramesh 2002). This problem can be traced to two attributes of knowledge that must be brought to bear on the development process: (1) fragmentation and (2) stickiness. First, the relevant knowledge is often dispersed among various project stakeholders, which makes its embodiment in the design of the software challenging (Ramesh 2002). Requirements, for example, are derived from expressed or inferred customer needs. The actual design of the software is based on technical knowledge about programming languages, methodologies, and software architecture applied to customer-driven requirements. While the former comes from the customer organization, the latter resides largely in the vendor organization. Second, much of the knowledge in software engineering is "sticky." Sticky knowledge refers to knowledge that is so complex and context-dependent that it is difficult to convey (DeSouza 2003; Polanyi 1966). Although some knowledge related to software development can be codified in documents, stored in artifacts such as mock-ups and prototypes, or embodied in development methodologies, skills, expertise, and perspectives are held in the minds of individuals, i.e., sticky. Sticky knowledge cannot be easily codified in documents and artifacts yet is critical for software development. It must be converted into declarative knowledge—facts and properties about objects, persons, events, and relationships—to be applied to software development activities (Robillard 1999). A commonly observed example of knowledge stickiness is the difficulty associated with accurately conveying customer needs for a system through formal requirements. In routine, well-understood design problems, stickiness is less problematic because formal requirements can faithfully define the problem space. However, when a project involves conceptual or process novelty, stickiness can be more problematic. This is because stickiness is higher when the vendor does not possess the complementary skills and understanding to be able to use the information that is conveyed by the customer or vice versa (von Hippel 1994). Stickiness can thus persist even when full cooperation exists between a customer and the vendor. Even if the customer possesses a mental model of a project's objectives, unless it is understood by the vendor, the prospect of embodying the customer's needs in the software is bleak.

The persistent challenges in software development—unmet user needs, requirements mis-matches, and systems that fail to meet customer expectations—thus arise from inadequate integration of the business knowledge and technical knowledge in the software development process. The importance of knowledge integration has implicitly been long recognized in the software engineering literature because many software engineering concepts such as information hiding, objects, patterns, functions, procedures, and modularity aim to guide or ease the processing of knowledge (Robillard 1999; Zweben et al. 1995). In summary, knowledge held by the customer and the vendor carries little value by itself unless it is embodied in the design and functionality of a software system. Knowledge integration is central to this process. We next discuss how knowledge partitioning—the dispersion of technical and business application domain knowledge across the customer and vendor organization—influences knowledge integration during software development.

Knowledge Partitioning

Knowledge partitioning refers to the pattern of distribution of technical and application domain knowledge across the customer and vendor organizations at the outset of a project. Knowledge can be symmetrically or asymmetrically partitioned. Knowledge is defined as *symmetrically* partitioned when it mirrors the partitioning of development activities between the customer and vendor organization, i.e., the customer organization does not possess detailed technical knowledge about the project and the vendor does not have detailed knowledge of the customer's business. Knowledge is defined as *asymmetrically* partitioned if the customer or the vendor knows more than their roles in the development process described below usually require. An example of this is when the customer understands the project's technical intricacies or the vendor deeply understands the customer's business domain.

Software Project Type	Vendor's knowledge about customer's business application domain	Customer's technical knowledge about the project	Knowledge partitioning
0. Non-optimal	**High**	**High**	←Symmetric
1. Routine	**Low**	**Low**	
2. Conceptual newness	**High**	**Low**	←Asymmetric
3. Process newness	**Low**	**High**	

Figure 1. Hypothesized Knowledge Partitioning Patterns

The salient knowledge partitioning scenarios are illustrated in Figure 1 and are discussed next. (Although we do not directly analyze the uncommon high-high symmetric case 0, we later discuss the implications of our results for it.)

Case 1: Routine Projects

Consider first the case where a software project is routine, i.e., it is neither conceptually new nor involves process newness (new methodologies, new development tools, or both). This case represents a reasonably definable problem space, i.e., one for which the initial state, the goal, and a set of possible operations to reach the goal from the initial state are available (Robillard 1999). A conventional black-box approach relying on the propositional representation of the design problem in formal project requirements is apt in this case. Here, the customer and vendor can agree on requirements that capture needs of the desired system with reasonable accuracy. Once the requirements are understood, the vendor can build a system that satisfies these requirements. Requirements then act as a black-box between the software and the customer's business needs. A rigorous requirements analysis process then becomes the key knowledge integration mechanism for mapping the customer's business needs to the design of the software. As long as these requirements capture the essence of the project's needs, there is little need for the customer to understand the technical details of the outsourced project or for the vendor to understand the intricacies of the customer's business application domain. This represents the ideal symmetrical case of knowledge partitioning where the vendor does not need high levels of knowledge about the customer's business nor the customer about the technical minutiae of the project.

However, the costliest and least remediable defects in software stem from misunderstood requirements (Cantor 2002). The black-box approach is unreliable for transferring knowledge about customer needs smoothly and completely to the vendor through formal requirements in projects involving newness. Newness introduces unpredictable interdependencies between the customer and vendor's activities related to the software project (Schulz 2001; von Hippel 1990). Interdependence refers to the probability that the completion of some development activities by the vendor will require related activities by the customer beyond just the requirements stage. When interdependencies cannot be fully anticipated, it is more difficult to accurately specify the business needs from which the preliminary solution is devised. Overlaps in vendor-customer knowledge then become necessary for integrating knowledge. In such scenarios, symmetric knowledge partitioning is suboptimal because formal requirements will likely provide an incomplete picture of customer needs to the vendor. The need for overlap necessitates an asymmetric knowledge partitioning pattern where the vendor or the customer has higher knowledge about the other's task domain. Depending on whether the customer or the vendor has more knowledge about the others' activity domain, two asymmetric knowledge partitioning patterns illustrated in cases 2 and 3 in Figure 1 can emerge. The conditions under which these become necessary are discussed next.

Case 2: Conceptually New Projects

When a project is conceptually new to the vendor, it is difficult to convey customer needs solely through a black-box, requirements-driven approach. Rarely can such the ideas behind conceptually new applications be captured as discrete, objective facts during the requirements elicitation process (Rowen 1990; Rus and Lindvall 2002). This is because conceptually novel projects must draw on knowledge from multiple technical and functional domains (Curtis et al. 1988). Moreover, stickiness of customer knowledge is higher when the vendor does not deeply understand the customer's problem space (as is the norm under symmetric knowledge partitioning). It then becomes difficult for the vendor to assess what subset of knowledge conveyed by the customer is relevant to the project. If the symmetric partitioning pattern described in case 1 persists, the vendor is likely to lack sufficient understanding about the customer's business, thereby encountering difficulties in integrating expressed customer needs in the development process. It is, therefore, difficult to encode the customer's needs for conceptually new software in the initial requirements. Conflicting or incomplete interpretations of a project's goals by the vendor can lead to design decisions that are incompatible with the customer's project objectives (Cooper 2000). To fully comprehend such needs, the vendor must possess more knowledge about the customer's business application domain than is necessary in routine projects. Further, reduced analyzability of the design problem space in conceptually new projects requires more customer-vendor coordination in the later (post-requirements) phases of the development process (Adler 1995). Therefore, in conceptually new projects, it is important for the vendor to have a more knowledge about the customer's business application domain. This asymmetric knowledge partitioning pattern is illustrated in case 2 in Figure 1.

Case 3: Projects Involving Process Newness

Process newness is defined as the extent to which a project involves development processes that are new to the vendor. Process newness arises from use of a software development methodology (e.g., agile programming or pair-programming) or software

development tools (e.g., an integrated object-oriented CASE tool) with which the vendor has limited prior experience. Process newness makes less predictable existing processes that might work well in routine software projects. In such cases, it is important for the customer to be able to distinguish effective from ineffective development processes (Cantor 2002), an ability that is less critical in projects using well-established software development processes. The customer's technical knowledge of the project allows comprehension of the development process. Without the relevant technical knowledge about the software, the customer is unlikely to successfully exploit opportunities to improvise the design because of an inability to interpret—or act on—information that might be provided by the vendor during walkthroughs, inspections, and user reviews. Therefore, the customer's technical knowledge helps the customer validate intermediate design artifacts throughout the development trajectory, which enhances knowledge integration. Therefore, in projects involving process newness, it is important for the customer to have a more technical knowledge than is ordinarily necessary.[1] This asymmetric knowledge partitioning pattern is illustrated in case 3 in Figure 1. These ideas are summarized in Figure 1 and are tested empirically for the first time in this study.

Integration Mechanisms

Mechanisms that reduce the barriers to problem-solving across the customer-vendor boundary can enhance integration of knowledge distributed across that boundary. Such integration mechanisms must support both explicit knowledge that can be captured in documents, reports, and code fragments as well as tacit knowledge that cannot be easily explicated. Five integration mechanisms have been *proposed* in previous research: (1) development tools that facilitate the coordination of various development activities, (2) close customer-vendor interactions, (3) up-front effort in designing the architecture of the system with the customer, (4) use of mature software processes, and (5) the vendor's collaborative experience with a customer.

1. *Development coordination tools.* Various coordination tools such as requirements managers, architectural modeling tools, test case development tools, configuration managers, and defect and change request tracking tools have been proposed to enhance coordination of development activities across project stakeholders and participant groups. Such tools allow project participants to coordinate development activities by providing access to work artifacts during the course of development (Rus and Lindvall 2002). Anecdotal evidence from leading adopters of such tools suggests that they can reduce design defects by as much as 40 percent (Ramasubramaniam and Jagadeesan 2002), enhance knowledge sharing across space and time (Ramesh 2002; Ramesh and Dhar 1992), expose design inconsistencies (Cantor 2002), and improve overall design quality (Rose 1998). We therefore expect that the use of development coordination tools will positively impact knowledge integration in the software development process.

2. *Customer-vendor interaction.* When knowledge about customer needs is complex and sticky, formal requirements are likely to capture only a subset of a project's true requirements. In such cases, ongoing vendor-customer interaction during the development process can help surface requirements that are not captured at the outset of the development process (DeSouza 2003; Jalote 2000). Previous studies suggest that such interactions can also serve to iteratively refine existing requirements and their mapping to the design of the software (Ellis and Tyre 2001; von Hippel 1994; Walz et al. 1993). Such interactions complement development coordination tools because they facilitate flow of sticky knowledge across the vendor-customer boundary that is less likely through coordination tools. We therefore expect that customer-vendor interaction enhances knowledge integration during the software development process.

3. *Architectural design effort*. Software can only be as good as its design. When more effort is invested in the initial conceptual design phase of a software development project, it is more likely that customer needs will be more accurately identified and mapped into a design that satisfies those needs (MacCormack et al. 2001). Therefore, the level of effort expended in the design of a system will enhance the level to which an understanding of the customer's needs is reflected in the software design.

4. *Maturity of software processes.* More mature development processes can better cope with unreliable preliminary information in the software development process (Sengupta and Abdel-Hamid 1996). The most widely used measure for software process capability is the vendor's capability maturity (CMM) level (Jalote 2000). All else being equal, the higher maturity of software processes used by a vendor, the higher will be the level of knowledge integration in the software development process.

[1]However, such knowledge also has a potential downside: A customer with in-depth technical understanding about the project is more likely to attempt to exert control on the development process and demand additional features and functionality, all of which can interfere with the development activities (Kirsch et al. 2002).

5. *Collaborative development experience.* Previous vendor development experience with a customer can provide the necessary understanding of the customer organization's norms, practices, and work practices. Such understanding will likely enhance knowledge integration.

Research Methodology

This study initially involved 232 projects in 232 different software development companies in Russia, Ireland, and India for external customers, all of which were American firms. A total of 818 potential organizations for the study were identified through three of the largest global software consortia in Russia, Ireland, and India. These respectively were the Russian National Software Development Alliance, member software firms in the Irish Investment and Development Agency (IDA), and National Association of Software and Service Companies (NASSCOM). Only companies that had significant partnerships with U.S. firms were chosen for this study to maintain comparability. Of these, 23 projects for which matched pair data sets could not be successfully collected were dropped. This left us with 209 projects for analyses.

Construct Operationalization

The questionnaire used existing measurement scales where possible. New measures were developed only when no measure existed for a construct. Prior research on knowledge management in software development has little empirical work, so new measures were developed for the following constructs: customer's technical knowledge, vendor's business application domain knowledge, and knowledge integration. These measures were developed based on detailed interviews and discussions with 19 software project managers (seven in Russia, six in the U.S., two in Ireland, and four in India) and seven academic experts. Each construct was measured using four to six questionnaire items. Most responses were measured using seven-point Likert scales or Guttmann scales and using objective data where possible. Traditional convergent validity and discriminant validity assessment procedures were used to refine the scales. The items retained for the analysis exhibited sufficient discriminant validity and construct scale reliability. Space limitations preclude a detailed discussion of the instrument, but the key measurement properties of each construct measure are summarized below. (Actual items are available from the author.)

Dependent Variable

1. Knowledge integration (*Know_Integ).* Six items based on prior work (Faraj and Sproull 2000; Walz et al. 1993), $\alpha = 0.74$, Likert scale.

Knowledge Partitioning Variables

1. Customer's technical knowledge (*Customer_TechKnow).* Six items, $\alpha = 0.91$, Likert scale.
2. Vendor's business knowledge *(Vndr_BusKnow).* Six items, $\alpha = 0.81$, Likert scale.
3. Project's process newness *(Proc_New).* Measure adapted from Takeishi (2002); four-item Guttman scale.
4. Project's conceptual newness *(Concept_New).* Measure adapted from Takeishi (2002); four-item Guttmann scale.

Integrating Mechanisms

1. Development coordination tools (*Coord_Tools*). Six-item Guttman scale that assessed the extent to which each of the following development coordination tools identified by Cantor (2002) was used in the project: requirements managers; architectural modelers; test automation tools; test case development tools; configuration managers; defect and change request tracking tools.
2. Customer-vendor interaction (*CV_Interaction*). Three-item, seven-point semantic differential scale adapted from Hansen (2002).
3. Architectural design effort (*Design_effort*). This scale was directly reused from MacCormack et al. (2001). The construct was measured as percentage of project hours allocated to design out of the total time spend on the following project activities: development, design, project management, and testing.
4. Maturity of software processes (*CMM_Level*). CMM level of the vendor organization.

5. Collaborative development experience (*CV_Experience*). Binary variable set to 1 if the vendor had prior development experience with the customer.

Data Collection

Data for the study were collected from key project managers in the vendor organizations and their customer-side liaison managers through a field survey. The key informant technique, which relies on a highly knowledgeable individual from each organization was used (Kumar et al. 1993). The data collection was done in two phases, which were preceded by the questionnaire development interviews discussed earlier. In the first phase, we asked the CEO or president of each software company to identify one major project that they recently completed for a U.S. customer. We sent a questionnaire to the lead manager of this project in each firm. In the second phase, we asked the key contact for each project to provide us objective archival data on the project. This was primarily related to the defects recorded at each stage of the development process, and the percentage by which the project exceeded the original schedule and budget. At this stage, we also asked the project manager to forward a shorter survey to the primary customer-side liaison in the customer organization. The purpose of this step was to collect data on the independent and dependent variables from different respondents in order to mitigate the threat of mono-methods bias. Only projects for which complete dyadic datasets were collected are included in the final analysis.

The response rates across the sample are summarized in Table 1. Given the sensitive nature of the data and the challenges associated with getting multiple assessments from both the customer and vendor organizations, the overall response rate of 28.4 percent compares favorably to typical field studies involving managers. To test whether the non-responding organizations were biasing our sample, we used follow-up calls to nine randomly selected non-responding vendors (three each in Ireland, Russia, and India). Most non-participating organizations declined to participate for lack of time, concerns about sensitivity of the data, or because they were no longer in the custom development business, thereby suggesting that nonresponse bias is not a persuasive threat to our findings.

Table 1. Response Rate

	Russia	Ireland	India	Full sample
Valid initial sample	176	183	459	818
Respondents	59	54	119	232
Response rate	33.5%	29.5%	25.9%	28.4%

Analyses and Results

Model Specification

Based on the theoretical model hypothesized in Figure 1, a stepwise regression model with three incremental equations was used to estimate the model. This specification begins with a baseline set of variables in equation (1), then adds equation (2) to the analysis to estimate the effects of knowledge partitioning on knowledge integration, and finally appends equation (3) to evaluate the effectiveness of the various integrating mechanisms for enhancing knowledge integration. The baseline model includes only *Customer_TechKnow, Vndr_BusKnow, Proc_New,* and *Concept_New*. This corresponds with the symmetrical knowledge partitioning scenario discussed previously in case 1 of Figure 1. The dependent variable to be estimated is knowledge integration.

$$\begin{aligned} Know\_Integ = {} & \beta_0 + \beta_1 Vndr\_BusKnow \\ & + \beta_2 Customer\_TechKnow \\ & + \beta_3 Proc\_New \\ & + \beta_4 Concept\_New + \varepsilon_1 \end{aligned} \quad (1)$$

Next, the contingent effects of the two types of knowledge and newness represented in the two asymmetric knowledge partitioning scenarios in Figure 1 are incorporated in the model. This can be formally specified by adding four interaction terms between the two types of knowledge and two types of newness variables to equation (1).

$$+ \beta_5 Customer\_TechKnow * Concept\_New + \beta_6 Customer\_TechKnow * Proc\_New + \beta_7 Vndr\_BusKnow * Proc\_New + \beta_8 Vndr\_BusKnow * Concept\_New + \varepsilon_2 \quad (2)$$

Finally, the five integration mechanisms that facilitate integration of business and technical knowledge during the development process can be formally incorporated by the following extension to equation (2).

$$+ \delta_1 Coord\_Tools + \delta_2 CV\_Interaction + \delta_3 Design\_effort + \delta_4 CMM\_Level + \delta_5 CV\_Experience + \varepsilon_3 \quad (3)$$

Descriptive Statistics

All projects in our study were custom application development projects. The average complexity of the projects was 2,241 function points (FPs). The duration of the projects ranged from one month to four years, with an average duration of about 11 months (s.d. 9.6 months). The average size of team dedicated to the project by the vendor consisted of 16 people. Of the vendors that had worked previously with the same customer on earlier projects, the average relationship history spanned about 3.6 years. This represented about 59 percent of the projects in the sample. The CMM levels of the vendors varied from none to 5. On average, the vendors in the study had been in the software development business for about 7 years, although this number varied from upstart companies to the oldest one at 47 years (s.d. 6.2).

Model Estimation

The stepwise regression model was estimated in three stages involving equations (1), (2), and (3). One-tailed T-tests were used because the hypothesized relationships are unidirectional. The results are summarized in Table 2 (all statistically significant results are shown in **bold**).

Equation (1) estimation. Equation (1) represents the baseline model that provides the empirical foundation for testing the knowledge partitioning patterns hypothesized in Figure 1. Three binary dummy coded variables Russia, Ireland, and India are included in the model to compensate for the national origin of each project data point that can confound the results. The coefficients for these dummy variables were statistically nonsignificant, suggesting that further analysis could proceed by pooling all projects since no significant differences in knowledge integration were caused by the national origins of the vendors. *Customer_TechKnow* and *Vndr_BusKnow* are nonsignificant as well, as expected for the first scenario in Figure 1. This is consistent with prior studies that suggest that the mere presence of the two types of knowledge by itself does not affect software development (Faraj and Sproull 2000). Similarly, *Concept_New* and *Proc_New* also have nonsignificant coefficients, indicating that the presence of conceptual or process newness in projects does not directly affect knowledge integration during software development.

Equation (2) estimation. In the next step, the four interaction terms for equation (2) were added to the baseline model represented by equation (1). The results of this regression run are summarized in the second "knowledge partitioning" column in Table 2.

- *Conceptually new software:* A positive and statistically significant effect of the β_8 term in equation (2) suggests that the vendor's knowledge of the business application domain knowledge significantly improves knowledge integration in software projects that are conceptually new. In such projects, the customer's technical knowledge about the software development process and technologies does not contribute to knowledge integration as indicated by a nonsignificant β_5 term.

- *Software involving process newness.* The interaction term represented by β_6 has a positive and statistically significant effect on knowledge integration. This result provides evidence that the customer's technical knowledge significantly improves knowledge integration in projects that involve process newness. The negative and significant effect of β_7 suggests that the vendor's business application domain knowledge does not have positive effect in this case.

These results provide evidence that (1) a vendor needs a high level of business domain knowledge in conceptually innovative software projects and (2) the customer needs a higher level of technical knowledge in projects that rely on new software development processes. This pattern of asymmetric knowledge overlaps was discussed earlier in Figure 1. Further, model (2) explains significantly more variance in knowledge integration as indicated by the ΔR^2 value in the second regression model.

Table 2. Results

	Equation 1 Baseline Model			Equation 2 Knowledge Partitioning			Equation 3 Integration mechanisms		
	Coefficient	Std Error	T-value	Coefficient	Std Error	T-value	Coefficient	Std Error	T-value
Constant (β_0)	—	0.24	16.34	—	0.43	7.70	—	0.55	4.95
Russia	–0.06	0.09	–0.89	–0.04	0.08	–0.78	–0.03	0.06	–0.60
India	0.04	0.14	0.33	0.03	0.13	0.24	–0.01	0.13	–0.05
Ireland	–0.01	0.17	–0.04	0.02	0.16	0.16	–0.03	0.17	–0.21
Vndr_BusKnow(β_1)	0.07	0.09	0.59	–0.19	0.31	–0.45	–0.33	0.32	–0.78
Customer_TechKnow(β_2)	–0.14	0.08	–1.19	**–0.94•**	**0.29**	**–2.17**	–0.81	0.29	–1.86
Proc_New(β_3)	0.01	0.06	0.08	0.13	0.12	0.52	0.25	0.13	0.99
Concept_New(β_4)	0.04	0.07	0.32	0.30	0.15	1.19	0.18	0.15	0.71
*Customer_TechKnow*Concept_New*(β_5)				0.10	0.11	0.19	–0.12	0.11	–0.22
*Customer_TechKnow*Proc_New* (β_6)				**0.89••**	**0.08**	**2.61**	**0.95••**	**0.09**	**2.57**
*Vndr_BusKnow*Proc_New*(β_7)				**–0.75••**	**0.09**	**–2.62**	**–0.59•**	**0.10**	**–1.96**
*Vndr_BusKnow*Concept_New*(β_8)				**0.91•**	**0.11**	**2.04**	**0.89•**	**0.11**	**1.97**
CV_Interaction (δ_2)							**0.23•**	**0.05**	**2.06**
Design_effort(δ_3)							0.03	0.01	0.29
CMM_Level(δ_4)							0.11	0.06	0.97
CV_Experience(δ_5)							0.17	0.12	1.54
Coord_Tools (δ_1)							–0.05	0.01	–0.38
R^2_{Adj}	2.4%			21.5%			29.1%		
ΔR^2	—			19.1%••			7.6%•		
F–change	—			4.51			1.74		

••p > 0.01 two–tailed test
•p > 0.05 two–tailed test

Equation (3) estimation. The third step assesses the effectiveness of various integration mechanisms. In this final step, equation (3) is added to the previous model and the entire model is reestimated. As shown in Table 2, the direction of the coefficients for all mechanisms except *Coord_Tools* was in the positive direction. This is consistent with our earlier predictions about their effect. Vendor-customer interaction was the only significant predictor, suggesting that close customer-vendor interactions are the single most reliable mechanism for integrating business application domain knowledge in the software design process. Recall that we hypothesized about the other integrating mechanisms based primarily on prior case studies or anecdotal evidence. There is clearly a need for finer grained empirical research to fully understand how they affect knowledge integration during software development. We also conducted tests to assess the threat of mono-methods bias but found no evidence that suggested its presence.[2]

[2]To assess the threat of mono-methods bias, we tested whether vendor assessments of knowledge integration were associated with customer assessments of software development efficiency and effectiveness. The extent to which a project is completed within the allocated resources was used to assess development efficiency. We regressed the percentage by which each project exceeded its original budget on knowledge integration ($\beta = –0.238$, T-value = –2.28, $p < 0.05$). We found a negative and statistically significant relationship between the two, suggesting that higher levels of knowledge integration are associated with higher development efficiency. Development effectiveness was measured using *customer* assessments of the following on a five-item, seven-point Likert scale: system reliability, implementation of functionality, meeting project objectives and functional requirements, and overall fit with customer needs ($\alpha = 0.91$; mean = 5.5; s.d. = 0.91). A positive and statistically significant relationship ($\beta = 0.24$, T-value = 2.84, $p < 0.01$) suggests that knowledge integration is positively associated with higher software development effectiveness. These tests suggest that mono-methods bias is not a pervasive threat to our findings.

Discussion and Implications

The results of this research provide insights into the conditions under which it is necessary for vendors and customers to possess knowledge of each others' specialized activities for successful software development. They also provide evidence for the desirable pattern of knowledge partitioning asymmetries when projects are conceptually novel or involve new development processes. Our results for equation (1) show that symmetrical partitioning of knowledge is optimal in routine software development. The relatively well-understood nature of project concepts and the relatively predictable interdependencies between customer needs and software design are conducive to communicating project needs through the requirements process. In this case, the customer requires little technical knowledge and the vendor requires limited knowledge about the customer's business domain.

However, it is important to have asymmetrically partitioned knowledge when a project is conceptually novel or involves processes newness. Asymmetry can take two forms: (1) the vendor has a higher level of business knowledge or (2) the customer possesses in-depth technical expertise. Overlaps in knowledge at the vendor-customer boundary differentially influence knowledge integration when one considers the extent to which a project involves conceptual newness and process newness. Our findings show that higher vendor-side knowledge is necessary in conceptually novel projects and higher customer-side knowledge is necessary when novel processes are involved. This is illustrated in Figure 1. When organizations do not consider the needed asymmetries in knowledge partitioning, they inadvertently set themselves up for failure. The results from equation (2) provide insights into when higher levels of technical knowledge in the customer organization and higher levels of knowledge about the customer's business application domain in the vendor organization benefit software development. The nature of optimal knowledge asymmetries varies depending on whether a project involves conceptual newness or process newness.

Vendor's business knowledge. The vendor's business application domain knowledge benefits software development when a project is conceptually new. However, this type of knowledge asymmetry can hinder knowledge integration when the project involves the use of new processes. In the case of process newness, it is important for the customer to have a higher level of technical knowledge. Although previous case studies have recommended that software development organizations should invest in increasing the business knowledge of software development staff (Walz et al. 1993), our findings suggest that investments in doing so are valuable only if a vendor routinely develops conceptually new applications. Educating a vendor about the customer's business application domain is not cost-free and usually increases the cost of the project (Walz et al. 1993). In routine applications, there are limited benefits in doing so. More surprisingly, if a vendor routinely develops applications involving new processes, investing in internally developing such knowledge can invoke negative consequences by *inhibiting* integration of their technical knowledge with the customer's business knowledge (as indicated by the negative sign associated with the β_7 term).

Customer's technical knowledge. Equation (2) also provides insights into when higher levels of technical knowledge in the customer organization benefit software development. Our results suggest that such knowledge is useful only when a project involves process newness. In all other cases, it is unnecessary—even undesirable—for the customer organization to possess high levels of technical expertise. In routine projects, technically knowledgeable customers are more likely to interfere with the development process and inadvertently hinder knowledge integration during the software development process, as hinted in prior information systems development research (see Kirsch et al. 2002). The negative and statistically significant coefficient for the β_2 term in equation (2) provides empirical support for this. In summary, high levels of technical knowledge in the customer organization are necessary only in the presence of process newness in a project. If customers or vendors assume that symmetric partitioning is unconditionally appropriate, they lose the opportunity to fully exploit available knowledge for developing an appropriate software solution.

Three important implications can be drawn from this finding for software practice. First, investments by software development organizations in enhancing the business expertise of developers are useful only if conceptually novel projects are routinely undertaken by a vendor. This is contrary to the popular belief that software professionals must have in-depth business expertise to successfully develop software. Second, vendors need to "educate" customers on the intricate technical details of a project only when it involves process newness. Under all other scenarios, it is unnecessary for customers to possess in-depth technical knowledge. Third, it is usually difficult for organizations to recognize the nature of vendor-customer interdependence at the outset of a new project. Our findings approach this problem from a different direction by providing a powerful approach for predicting the kind of interdependence that a customer and vendor will face based on the conceptual and process newness of the project (which are relatively easier to judge). Based on this, the needed knowledge overlaps can be recognized at the outset of a project and the extent to which they characterize a given vendor-customer pair. The required knowledge partitioning patterns based on these results are summarized in Figure 2, which shows the scope of knowledge that a vendor and customer need to cover in each case.

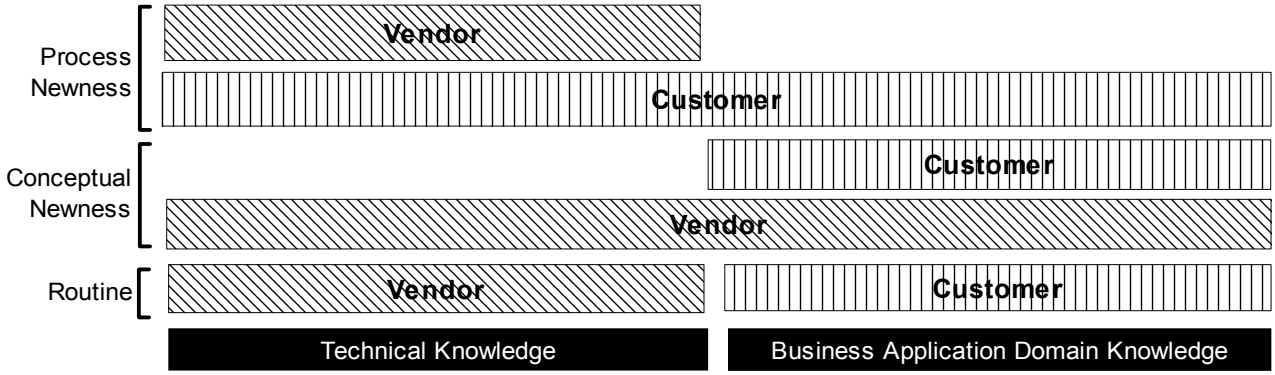

Figure 2. Knowledge Partitioning Patterns in Routine, Conceptually New, and Procedurally New Projects

Our results for equation (3) also suggest that frequent communication and close working relationships between the vendor and customer are the most reliable of mechanisms for integrating knowledge and managing vendor-customer interdependencies during software development. It is noteworthy that the projects in the study were drawn from firms in Russia, Ireland, and India, where the majority of global outsourced software development occurs. This increases the generalizability of our findings.

Conclusions

The objective of this study was to assess when it is necessary for vendors and customers to possess detailed knowledge of each others' domains for successful outsourced software development. We developed a theory-driven predictive model for assessing the conditions that require overlap in knowledge at the vendor-customer boundary. Traditionally, software development has relied on a formal requirements-driven black-box approach in which such overlap is considered unnecessary and symmetric knowledge partitioning is optimal. We also explored whether the asymmetry should be skewed toward the customer or vendor in the presence of conceptual and process novelty. The model was tested using data on 209 software projects outsourced by American firms to 209 software development companies in Russia, Ireland, and India.

The study makes three important contributions. First, we show that it is more important for a vendor to possess higher levels of business application domain knowledge in conceptually new projects and for the customer to have a higher level of technical knowledge when the project involves new development processes. On the other hand, such overlaps are unnecessary in routine software projects. This finding about the contingencies under which vendor-customer knowledge overlaps enhance knowledge integration—and in turn software development efficiency and effectiveness—is a unique contribution to the literature. Second, we show that close customer-vendor interactions during the development process enhance knowledge integration. Third, we show that there is a downside for the vendor to invest in increasing knowledge about the customer's business application domain in the presence of process newness but it is helpful in conceptually new projects.

Clearly, managing the fluid nature of knowledge boundaries poses special challenges for software development organizations. An appreciation of the contingencies described here can allow software development organizations to further leverage their most underutilized, weightless wealth: Their knowledge assets.

References

Adelson, B., and Soloway, E. "The Role of Domain Experience in Software Design," *IEEE Transactions on Software Engineering* (SE-11:11), 1985, pp. 1351-1360.

Adler, P. S. "Interdepartmental Interdependence and Coordination: The Case of the Design/Manufacturing Interface," *Organization Science* (6:2), 1995, pp. 147-167.

Armour, P. "A Case for a New Business Model: Is Software a Product or a Medium?," *Communications of the ACM* (43:8), 2000, pp. 19-22.

Cantor, M. *Software Leadership: A Guide to Successful Software Development*, Addison-Wesley, Indianapolis, 2002.
Constantine, L., and Lockwood, L. "Orchestrating Project Organization and Management," *Communications of the ACM* (36:10), 1993, pp. 31-43.
Cooper, R. "Information Technology Development Creativity: A Case Study of Attempted Radical Change," *MIS Quarterly* (24:2), 2000, pp. 245-276.
Curtis, B., Krasner, H., and Iscoe, N. "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM* (31:11), 1988, pp. 1268-1287.
DeSouza, K. "Barriers to Effective Use of Knowledge Management Systems in Software Engineering," *Communications of the ACM* (46:1), 2003, pp. 99-101.
Ellis, S., and Tyre, M. "Helping Relations between Technology Users and Developers: A Vignette Study," *IEEE Transactions on Engineering Management* (48:1), 2001, pp. 56-69.
Faraj, S., and Sproull, L. "Coordinating Expertise in Software Development Teams," *Management Science* (46:12), 2000, pp. 1554-1568.
Hansen, H. "Knowledge Networks: Explaining Effective Knowledge Sharing in Multiunit Companies," *Organization Science* (13:3), 2002, pp. 232-248.
Jalote, P., *CMM in Practice,* Addison-Wesley, Indianapolis, 2000.
Kirsch, L., Sambamurthy, V., Ko, D., and Purvis, R. "Controlling Information Systems Development Projects: The View from the Client," *Management Science* (48:4), 2002, pp. 484-498.
Kumar, N., Stern, L.W., and Anderson, J.C. "Conducting Interorganizational Research Using Key Informants," *Academy of Management Journal* (36:6), 1993, pp. 1633-1651.
MacCormack, A., Verganti, R., and Iansiti, M. "Developing Products on Internet Time: The Anatomy of a Flexible Development Process," *Management Science* (47:1), 2001, pp. 133-150.
Polanyi, M., *The Tacit Dimension*, Doubleday, New York, 1966.
Ramasubramaniam, S., and Jagadeesan, G. "Knowledge Management at Infosys," *IEEE Software* (May/June), 2002, pp. 53-55.
Ramesh, B. "Process Knowledge Management with Traceability," *IEEE Software*, May/June 2002, pp. 50-55.
Ramesh, B., and Dhar, V. "Supporting Systems Development by Capturing Deliberations During Requirements Engineering," *IEEE Transactions On Software Engineering* (18), 1992, pp. 498-510.
Robillard, P. "The Role of Knowledge in Software Development," *Communications of the ACM* (42:1), 1999, pp. 87-92.
Rose, T. "Virtual Assessment of Engineering Processes in Virtual Enterprises," *Communications of the ACM* (41:12), 1998, pp. 45-52.
Rowen, R. "Software Project Management under Incomplete and Ambiguous Specifications," *IEEE Transactions on Engineering Management* (37:1), 1990, pp. 10-21.
Rus, I., and Lindvall, M. "Knowledge Management in Software Engineering," *IEEE Software* (May/June), 2002, pp. 26-38.
Sanchez, R., and Mahoney, J. "Modularity, Flexibility, and Knowledge Management in Product Organization and Design," *Strategic Management Journal* (17:1), 1996, pp. 63-76.
Schulz, M. "The Uncertain Relevance of Newness: Organizational Learning and Knowledge Flows," *Academy of Management Journal* (44:4), 2001, pp. 661-681.
Sengupta, K., and Abdel-Hamid, T. "The Impact of Unreliable Information on the Management of Software Projects: A Dynamic Decision Perspective," *IEEE Transactions on Systems, Man, and Cybernetics* (26:2), 1996, pp. 177-189.
Takeishi, A. "Knowledge Partitioning in the Interfirm Division of Labor: The Case of Automotive Product Development," *Organization Science* (13:3), 2002, pp. 321-338.
von Hippel, E. "'Sticky Information' and the Locus of Problem Solving: Implications for Innovation," *Management Science* (40:4), 1994, pp. 429-439.
von Hippel, E. "Task Partitioning: An Innovation Process Variable," *Research Policy*, 1990, pp. 407-418.
Walz, D., Elam, J., and Curtis, B. "Inside a Software Design Team: Knowledge, Sharing, and Integration," *Communications of the ACM* (36:10), 1993, pp. 63-77.
Zweben, S. H., Edwards, S. H., Weide, B. W., and Hollingsworth, J. E. "The Effects of Layering and Encapsulation on Software Development Cost and Quality," *IEEE Transactions on Software Engineering* (21:3), 1995, pp. 200-208.