Knowledge in Software-Maintenance Outsourcing Projects: Beyond Integration of Business and Technical Knowledge

Oliver Krancher
University of Bern
oliver.krancher@iwi.unibe.ch

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

Abstract

Knowledge processes are critical to outsourced software projects. According to outsourcing research, outsourced software projects succeed if they manage to integrate the client’s business knowledge and the vendor’s technical knowledge. In this paper, we submit that this view may not be wrong, but incomplete in a significant part of outsourced software work, which is software maintenance. Data from six software-maintenance outsourcing transitions indicate that more important than business or technical knowledge can be application knowledge, which vendor engineers acquire over time during practice. Application knowledge was the dominant knowledge during knowledge transfer activities and its acquisition enabled vendor staff to solve maintenance tasks. We discuss implications for widespread assumptions in outsourcing research.

1. Introduction

Many western businesses outsource software work to access scarce skills and to reduce costs [1, 2]. While various software tasks may be outsourced, software maintenance—modifications to a software after its first deployment—is particularly popular given that up to 80% of software costs arise during maintenance [3]. Among the key success factors of software-maintenance outsourcing (SMO) is knowledge transfer to the vendor [1, 4, 5], i.e. the process through which the vendor acquires the knowledge to maintain the software. This may not surprise given the high cognitive demands of software maintenance work [6, 7].

Although knowledge is central in SMO, the literature has developed conflicting views on what knowledge is transferred or integrated in SMO. Much outsourcing research has relied on a two-domain knowledge taxonomy advocated by Tiwana [8]. In this taxonomy, knowledge is distinguished into business knowledge, which is usually held by the client, and technical knowledge, which is usually held by the vendor [1, 8]. It is often assumed that software projects are successful if they manage to integrate these distributed knowledge domains across company boundaries, either by transferring knowledge to the other party or by substituting the need for knowledge transfer by direction, e.g. through detailed design documents [1, 8, 9]. Tiwana proposed the two-domain taxonomy for new software development, but much subsequent outsourcing research extended it to outsourced software projects [10-12], which may partially or primarily be software maintenance. However, the two-domain view contrasts with software-maintenance research. Software maintainers rely on software-specific mental schemas to comprehend the often enormous amounts of software code [6, 13]. High-performing maintainers are therefore distinguished by their application knowledge (i.e. knowledge about the software application), which they have acquired through years of software-specific experience [14, 15]. Yet, application knowledge is not reflected in the established two-domain view.

If application knowledge is central in SMO, then this has implications for established views of how knowledge is exchanged. It challenges the view of software work as integration of the specialized knowledge that is core to the task of each party. Vendors may initially hold specialized technical knowledge to design and develop software and clients may initially hold specialized business knowledge to specify their requirements. But vendors initially lack knowledge of client-specific software applications although this knowledge is core to their maintenance task. Vendors then need not only to integrate knowledge peripheral to their tasks, but also to acquire knowledge core to their tasks, notably application knowledge. Better understanding knowledge domains in SMO would thus help better understand knowledge processes.

This paper aims to shed empirical light on the knowledge domains involved in SMO. We present data from six transitions of software-maintenance work to Indian vendors. The data support the essential role of application knowledge. Hence, the established two-domain taxonomy fails to grasp the major knowledge domain in all six cases. The results imply that knowledge processes in SMO are not only knowledge integration, but also expertise acquisition. We call for research that is more sensitive to the knowledge differences between maintenance and development.
2. Theoretical framework

Much outsourcing work relies on assumptions about the knowledge involved in software work. We next present the prevailing two-domain taxonomy. We then describe an alternative three-domain taxonomy, which includes application knowledge, and we review information-processing theories to explain why application knowledge is critical in SMO.

2.1. The two-domain taxonomy

According to the two-domain taxonomy, knowledge can broadly be distinguished into two categories: business knowledge and technical knowledge [8, 16]. Business knowledge is “knowledge about the customer’s business processes, business rules, activities, stakeholder needs, and the customer’s business objectives for the software” [16, p. 900]. Technical knowledge is 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)” [16, p. 900]. Many outsourcing studies have explicitly or implicitly embraced this taxonomy, assuming that projects need to integrate these two types of specialized, dispersed knowledge between client and vendor [e.g. 10, 12, 17, 18].

2.2. An alternative three-domain taxonomy

Although the two-domain taxonomy is the dominant view in outsourcing work, two studies deviate from it. Chua and Pan [19] describe a case of captive offshoring, in which application knowledge was particularly difficult to transfer to the delivery team. They defined application knowledge as knowledge about the software application, its structure, functionality, and behavior [19, 20]. Similarly, Gregory et al. [21] found that knowledge about the client’s systems and technology landscape (functional knowledge in their terms) was among the most difficult types of knowledge to transfer to the vendor. These observations suggest that application knowledge may merit separate attention. Unlike business knowledge, it is germane to the vendor’s task of maintaining the software. Unlike technical knowledge, it is often largely client specific, in particular if the software has been custom developed or highly customized. It is client specific because applications reflect the wealth of situated attempts that have been made over the life of an application to match technology affordances with business needs.

Support for a distinguished role of application knowledge comes from software maintenance studies. They see application knowledge as software-specific mental schemas of how a given software operates. It is thus distinct from the more generic technical knowledge, which is not bound to a particular software product [6]. Maintainers acquire software-specific mental schemas through software-specific experience, one of the strongest predictors of maintenance outcomes [14, 15, 22]. For instance, in one study, teams with less than three years of software-specific experience introduced about three times as many errors to the software than the more experienced teams, irrespective of their technical knowledge [22]. The key role of application knowledge has led software maintenance researchers seek explanations for this finding in information-processing theories from cognitive psychology.

2.3. Information-processing theories

Information-processing theories are a family of theories from cognitive psychology that share assumptions of human information processing. They assume that humans process information in the interaction of a severely limited working memory and a virtually unlimited long-term memory [23, 24]. This helps explain the role of domain-specific expertise such as software-specific knowledge. Experts in a domain hold well-developed and largely tacit schemas in their long-term memories that enable them to aggregate information to higher-order chunks. Chunking thus reduces the cognitive loads on their working memories and enables the performance of more complex tasks [25, 26]. Experts acquire schemas through years of practice in the domain [27]. In contrast, novices lack powerful schemas and thus the ability to chunk information. Their working memories are quickly overloaded by problems in which they are to combine more than about three information elements. This is why experts can solve complex problems in their domain, while novices fail even if all information is available to them [25, 26].

Although some work criticized information-processing theories for imposing too simplistic machine metaphors on human knowledge work [28, 29], information processing research has thriven in the past two decades, seeking to explain human behavior in complex rather than simple task domains [30, 31]. Among the most prominent theoretical developments within the information-processing framework are the expertise literature [27] and cognitive load theory [26, 32]. The expertise literature suggests that years of practice in a domain are essential for superior performance in the domain [27] and that the resulting expertise is much more limited to the boundaries of the domain than often assumed [33]. Cognitive load theory seeks to explain learning in complex task domains as a function of cognitive load, the mental demands that a given
task imposes on a given learner. A robust finding is that novices are likely to learn ineffectively unless high cognitive loads are prevented by strategies such as simple-to-complex sequencing of tasks, using simplified task types such as worked examples, and providing supportive information [32, 34].

2.4. Information-processing theories and outsourcing research

While information-processing theories may be useful to understand the role of application knowledge in SMO, they contrast with views established in outsourcing research. Information-processing theories suggest that vendor staff need to acquire substantial application expertise through practice before they can independently maintain the software [32, 34]. In contrast, many outsourcing studies do not see expertise acquisition, but knowledge integration at the heart of outsourced software projects. They assume that clients and vendors initially have specialized expertise in their task domains, but that their integration is difficult [1, 16, 35].

The confusion might arise from underappreciated differences between software tasks. The two-domain taxonomy and the knowledge integration paradigm have originally been suggested for new-software development [16]. In new-software development, application knowledge may play a marginal role since the application is yet to be built. Conversely, in software maintenance, application knowledge may be critical because engineers need to comprehend the often enormous existing software applications. This difference has, however, not been observed in outsourcing research. Two interpretations may be offered for this. First, the differences between new-software development and software maintenance have vanished in outsourcing because vendors apply superior knowledge management practices such as knowledge codification to reduce or eliminate their dependence on application-specific experience [35, 36]. As a result, SMO may have become primarily a knowledge integration problem. Second, application experience continues to be vital in SMO and outsourcing research can be strengthened by acknowledging the key role of it in SMO. We next present data to support the second interpretation.

3. Methods

We conducted a multiple-case study of six SMO transitions to vendor on-site coordinators. The case study method promised insight into the importance of knowledge domains in real-life settings across a set of projects [37]. It also allowed observing changes in maintenance performance over time and associating them with the accumulation of experience in different domains. We observed projects during on-site transitions, the phase at the outset of projects during which vendor on-site coordinators are present at the client site for knowledge transfer. We assumed that social interaction and experiential learning during transition grant insight into the knowledge domains involved in SMO.

Although the six cases referred to the same Swiss Bank as a client, they were selected to allow variation in types of software, the vendors involved, knowledge transfer approaches, and success. The cases spanned different software systems such as custom-developed data warehousing applications (cases 1, 2, 3), banking software packages (cases 4, 5), and custom-developed transactional systems (case 6). Three major Indian vendors were involved, allowing some insight into whether and how their knowledge management practices altered the required knowledge domains.

Data were collected through semi-structured interviews, observation of knowledge-sharing sessions, and archival data. The data were collected real-time [38]. Thirty-eight interviews were held with vendor engineers (VEs), subject-matter experts (SMEs) such as client engineers and engineers of former vendors, and client managers (CMs). Recall bias was mitigated by multiple interviews with the same participants. Open interview questions were intended to have the participants talk about knowledge transfer activities such as formal sessions, document study, informal help-seeking, and work on tasks, and the knowledge involved in these activities. They were also asked how difficult they perceived the tasks. Interview and observation transcripts amounted to 167,626 words or 593 pages. Archival data included project documents such as requirements specifications, design documents, and software documentation, amounting to 570 pages.

The data were subject to three analyses. First, the data were coded in NVivo 9. The coding categories included inductively derived knowledge subdomains such as a particular business concept, a particular application component, or a particular technical skill. These subdomains were then aggregated to the knowledge domains technical knowledge, business knowledge, and application knowledge. The following quote may help illustrate the process. Asked why the VE was unable to solve a particular task, an SME said:

“We are very much driven by metadata and that's sometimes quite complex. ... [The VE] struggled to interpret the results from the tests. ... It’s not so much about the logic, it’s about data content. We have over 100 metadata tables with sometimes more than 1 million entries. Explain 20 metadata tables with 10,000 entries each (laughing).” (SME 1, case 1)

The statement suggests that knowledge about the metadata was required for this task. The statement was
therefore coded to a knowledge subdomain metadata. According to the statement, the VE struggled to make sense of the metadata in this software application rather than of the technical concept of metadata or the business logic embedded into the metadata (“it’s not so much about logic”). We thus related the node metadata to application knowledge. More examples are given in Appendix A. The coding process resulted in 1163 coding instances referring to 357 knowledge subdomains (such as metadata). We calculated the counts of coding instances in each of the domains as one indicator of their relative importance [39, p. 213].

A second analysis examined to what extent experience in different knowledge domains was associated with the cognitive loads on VEs during practice. Cognitive overload has been reported as a major problem in maintenance transitions [19, 40]. We assumed that, the more important a knowledge domain is, the more would experience in the domain relieve the cognitive loads on VEs. To grasp the experiences relevant for a task, we built a matrix of knowledge subdomains and tasks, showing what knowledge subdomains were relevant for each of the 64 learning tasks, on which VEs worked during transition. Extracts of the matrix were validated by participants. The matrix allowed calculating the amount of prior experience in each knowledge subdomain that was relevant for a particular task. Consistent with learning curve research, we used logarithms of the numbers of prior task experiences in a subdomain [14] and averaged these values for all subdomains that were relevant for a given task as a predictor of cognitive load. Moreover, we coded cognitive load on a five-point scale based on statements of perceived difficulty, mental effort, and task performance [41]. Coding examples are given in Appendix A. We also coded other constructs of cognitive load theory such as task type simplification, supportive information, and task complexity. Two coders were involved in coding cognitive load, task type simplification, supportive information, and task complexity. Reliability analysis yielded good results with Cronbach’s alpha above .9 or Cohen’s Kappa above .7 for all constructs. The coding scheme is available on request.

These quantified data allowed estimating two fixed-effects models [42, p. 359] that regressed the cognitive load associated with a particular learning task on the amount of experience in different domains. Model 1 included only business and technical experience, consistent with the two-domain model established in outsourcing research. Model 2 additionally considered application experience. Both models included further predictors according to cognitive load theory as controls and a random intercept to account for effects at the transitions level given the nested nature of the data. The predictors of cognitive load were all in the expected direction, supporting the validity of the model. Analyses suggested that autocorrelation (insignificant rho) and multicollinearity (VIF below 2.1) were no major concerns. The key results from the regressions were the regression coefficients associated with experience in the different knowledge domains.

Finally, we compared the results from the preceding two analyses with the participants’ perspectives on the relative importance of knowledge domain. We also related the rationales provided by the participants to explanations in the literature such as information-processing theories and knowledge integration frameworks. We deliberately triangulated analysis strategies to increase the internal validity of the study [37], assuming that weaknesses of one strategy may be partially offset if results are consistent across strategies.

4. Results

The results from the three analysis strategies consistently suggest that application knowledge was the most important knowledge domain in six of the six cases.

4.1. Counts of coding instances

Table 1 shows the counts of coding instances per knowledge domain and case. The figures show that all three knowledge domains surfaced in all six transitions, but that they were not similarly salient in the data. In each case, application knowledge was the domain coded most frequently with an average of 56% of knowledge domain coding instances. Put differently, the two-domain taxonomy endorsed by much outsourcing work grasped 44% of the knowledge statements in our data.

<table>
<thead>
<tr>
<th>Case</th>
<th>Technical Knowledge</th>
<th>Business Knowledge</th>
<th>Application Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>44 (11%)</td>
<td>50 (13%)</td>
<td>294 (76%)</td>
</tr>
<tr>
<td>Case 2</td>
<td>58 (26%)</td>
<td>58 (26%)</td>
<td>106 (48%)</td>
</tr>
<tr>
<td>Case 3</td>
<td>50 (26%)</td>
<td>53 (28%)</td>
<td>87 (46%)</td>
</tr>
<tr>
<td>Cases 4, 5</td>
<td>21 (15%)</td>
<td>38 (28%)</td>
<td>77 (57%)</td>
</tr>
<tr>
<td>Case 6</td>
<td>40 (18%)</td>
<td>61 (27%)</td>
<td>126 (56%)</td>
</tr>
<tr>
<td>Average</td>
<td>19%</td>
<td>24%</td>
<td>56%</td>
</tr>
</tbody>
</table>

Although the count data may give one perspective on the relative importance of knowledge domains, the counts might be affected by the informants’ tendencies to selectively report on certain domains or by the coding procedure. The analysis reported next is less sensitive to these issues because it derives importance from correlations with outcomes.

1 Case 4 and 5 referred to transitions of the same software application to two VEs. The table shows the sums for both transitions.
4.2. Predicting cognitive load

Our second analysis examined how the accumulation of experience in each of the three knowledge domains helped the VEs reduce cognitive load. We assumed that, the more important a knowledge domain is, the more should experience gains in the domain relieve cognitive load. This analysis thus focused on knowledge acquisition rather than on knowledge integration. Table 2 shows the result of two fixed-effects models that regress cognitive load on experience in different knowledge domains, and on controls established in cognitive load theory [26, 32, 34]. Model 1 includes only business and technical experience (two-domain taxonomy). Model 2 additionally considers application experience (three-domain taxonomy). The models rely on 64 data points of 64 learning tasks on which the VEs worked during transition.

Table 2. Predicting cognitive load

<table>
<thead>
<tr>
<th>Predictor</th>
<th>Model 1</th>
<th>Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\beta$ (std. e.)</td>
<td>$t$-value</td>
</tr>
<tr>
<td>Intercept</td>
<td>.32 (.32)</td>
<td>.99</td>
</tr>
<tr>
<td>Technical Ex.</td>
<td>-.38 (.11)</td>
<td>-3.33**</td>
</tr>
<tr>
<td>Business Ex.</td>
<td>-.43 (.14)</td>
<td>-3.04*</td>
</tr>
<tr>
<td>Application Ex.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Controls:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coordinative Complexity</td>
<td>.39 (.09)</td>
<td>4.51***</td>
</tr>
<tr>
<td>Supportive Information</td>
<td>-.20 (.11)</td>
<td>-1.93</td>
</tr>
<tr>
<td>Task Type Simpl.: Partial</td>
<td>-.39 (.23)</td>
<td>-1.70</td>
</tr>
<tr>
<td>Task Type Simpl.: Full</td>
<td>-.84 (.32)</td>
<td>-2.65</td>
</tr>
</tbody>
</table>

(*$p = .10$, **$p = .05$, ***$p = .01$, ****$p = .001$, n = 64, dependent variable: cognitive load, random constant term, REML method, unit of analysis: learning task, all variables have been standardized except for the dichotomous variables that capture task type simplification)

The regression coefficients given in Table 2 indicate how changes in experience in different knowledge domains over time were related to changes in cognitive load over time. In model 1, both technical and business experience were statistically significantly related to cognitive load. When application experience was added in model 2, it was most strongly and significantly related to cognitive load. Technical experience was still significantly related to cognitive load, although the correlation was weaker than for application experience. Conversely, business experience was not statistically significantly related to cognitive load any more. A comparison of the two models yielded a log likelihood difference of 7.03, suggesting that model 2 is preferred to model 1 at $p = .01$ ($11.99 > \chi^2(1)$ with $\chi^2(1) = 6.64$).

Several findings flow from these results. They suggest that the VEs acquired significant knowledge during practice and that it was in particular the acquisition of application knowledge that made tasks feasible towards the ends of transitions. This strongly corroborates the preceding analysis of coding counts. Again, the established two-domain model would have missed an important, if not the most important, facet of the knowledge transfer, as indicated by the model comparison and the size of the regression coefficients. The significant coefficient for technical experience may surprise in light of prevailing assumptions. Vendors are assumed to bring superior technical knowledge into the projects. However, if VEs are experts in the technical domains of the tasks, why should their gains in technical knowledge during the relatively short transitions make a significant difference? We will turn to this question in the next subsection. The regression coefficients associated with business experience also deserve attention. While business experience is significantly related to cognitive load if application experience is ignored (model 1), the correlation becomes weak and insignificant if application experience is included (model 2). A possible interpretation is that VEs are more concerned with how business logic is implemented in a given software than with the business logic itself. Business knowledge may appear important for maintenance outcomes if the established two-domain taxonomy is endorsed (model 1). However, this observation can turn out to be at least partially deceptive if the three-domain taxonomy is used. The minor role of business experience is in line with Tiwana’s [8] assertion that projects do not demand major business knowledge transfer to the vendor unless they revolve around conceptually highly novel business requirements. The projects involved highly incremental software modifications that may not be considered novel.

4.3. The participants’ perspectives

The study participants’ perspectives shed further light on the relative importance of knowledge domains in the cases. They largely support the key role of application knowledge acquisition and they provide context for why different knowledge domains were of different importance. The participants’ perspectives also give some insight into whether and how knowledge codification practices affected knowledge transfer.

4.3.1. Application knowledge. In particular the four cases of custom-developed software abound with statements of difficult application knowledge acquisition. In these cases, the VEs often depended on the help by SMEs on application issues even later during transition, when knowledge gaps in other domains were not very salient any more. For instance, in case 1, the VE was assigned a modification request during the
third month of the transition. He struggled because he had little experience in the application component:

“This task is very low-level. It is about a particular table and how it is loaded.” (VE, case 1) – “If you had plenty of time and nobody available to help you, could you solve this request based on the source code?” (Interviewer) – “That would take a lot more time because coding is very complex. Doing that alone is very tough, almost impossible without any help.” (VE, case 1)

The statement suggests why application knowledge was so central for the task. Although the VE did not require significant business knowledge for the task (“very low-level”), although he impressed client SMEs with his technical knowledge, and although he had access to the source code of the software, he found it “almost impossible” to comprehend the software behavior that results from the source code. This aligns well with the interpretation suggested by the software maintenance literature [6] and information-processing theories [26]. Having little experience in the software component related to this task, he lacked domain-specific mental schemas to aggregate large amounts of source code to higher-order chunks. His working memory was therefore overstrained by the combinatorial complexity of the maintenance problem, making problem-solving tedious. Notably, this appeared to be related neither to business nor to technical knowledge.

We made similar observations in other tasks of this transition and in the other transitions of custom-developed software. The informants agreed that it was the lack of application expertise that most constrained the VEs’ performance:

“[When you are experienced in a system and something happens,] you know where this can happen because you know the application. Suppose you get data here and you know these data are coming from that table, and that table is also used in another application. [...] For [the VE] it will be a little bit difficult in the beginning. (SME 1, case 2)

“Application knowledge is vast.” (SME, case 3)

“[Application knowledge] is the main thing to gain” (VE, case 3)

“I had a lot of questions: Why something is there? Why do we get the data through this interface? Why are we feeding the data to this interface?” (VE, case 6)

Code comprehension was less an issue in the cases of packaged software. In these cases, the VEs could use their application expertise from prior projects to comprehend the client-specific aspects of the software:

“I know how [the software package] works, but then set-ups can slightly change like pre-processing and post-processing [of data]. [...] That knowledge is what you have to gather.” (VE, case 4)

“My mind could easily map what the difference [to other implementations of the same software package] is. [...] When I had first worked on this software package, it had been like working on a blank sheet of paper. [...] But if I have been through something, it always stays in the memory. My subconscious always has some images which never get lost.” (VE, case 5)

The statements suggest that the VEs could use the schemas from prior projects with the same software package to aggregate information about client-specific aspects of the software implementation. Unlike in custom-developed software maintenance, VEs may thus draw on prior application expertise in software package maintenance. This may make it easier to acquire the client-specific facets application knowledge. The difference between custom-developed and packaged software cases thus supports the central role of application knowledge for maintenance outcomes.

4.3.2. Technical knowledge. The participants’ perspectives also grant some insight into the reasons for the surprisingly strong correlation of technical experience and cognitive load. All VEs had four or more years of prior experience in the technology domains of their tasks such as data warehousing or host environments. They were also considered as technically highly proficient by client SMEs such as in this statement:

“[The VE] has a very firm grasp of all the technologies involved.” (SME 2, case 2)

It is thus unlikely that marginal improvements in generic technical skills during the few months of transitions caused significant decreases in cognitive load. Indeed, very few statements indicate that VEs depended on SME help on technology issues.

This is, however, different for one facet of technical knowledge: software process knowledge [16], i.e. knowledge about the software-maintenance processes in the projects. The VEs initially needed to learn the client’s software-maintenance processes, such as document templates, tools, and deployment processes:

“[The VE had to learn] what to do to properly process a business request. There is a realization specification. There are templates. Where are they? What to write where? [...] Then testing. How to fill the tool with life? What is the process?” (SME 2, case 1)

Although VEs needed to learn about processes, they did not perceive this knowledge as difficult. Moreover, the SMEs soon attested that the VEs had acquired sufficient process knowledge when VEs still struggled with application knowledge. For instance, while the VE in case 1 was overstrained by the complexity of the software application in month 3, he commented on software process knowledge:

“Process things are now clear to me.” (VE, case 1)

The VEs and SMEs in the other cases also perceived software process knowledge as less difficult:

“[The process is] clear, there is no confusion. Now
4.3.3. Business knowledge. The qualitative data suggest that business knowledge was less important because requirements engineers of the client were present in five of the six cases, as explained by one VE:

“I don't have much difficulty in grasping these tools and processes” (VE, case 3)

“Process knowledge is always a little bit simpler.” (SME, case 3)

“He needs to know where to test what and that there are requirements documents and solution descriptions. [...] He now knows all this.” (SME, case 6)

Information-processing theories may explain why software process knowledge was easier to acquire than application knowledge. Information about software processes may often be understood in isolation. For instance, the steps for filling a peer-review form can be understood without simultaneously understanding the steps for checking in the software into a code repository or documenting test results. In contrast, information elements about the software application, such as lines of code or data, may often need to be combined to enable VEs to comprehend the software. However, simultaneously attending to many information elements quickly exceeds working memory capacity.

In sum, the correlation between technical experience and cognitive load may thus reflect that VEs initially need to familiarize themselves with the client’s software processes. But the acquisition of software process knowledge by the VEs is unlikely to be a performance bottleneck. By and large, this aligns with the established view that the transfer of technical knowledge to the vendor is not a major concern [8].

4.4. Synthesis

In sum, the three analyses delineate a highly consistent picture of the knowledge involved in the transitions. While all knowledge domains surfaced in all transitions, application knowledge was the most important. VEs were able to independently solve tasks only after they had acquired working experience in the components of the application to which the task referred. They struggled much more to comprehend the material at the core of their task domain (i.e. the application) than to access or assimilate peripheral knowledge such as business knowledge.

5. Discussion

Although knowledge processes are central to SMO, literatures have developed conflicting views on what knowledge is involved and how knowledge is exchanged. Whereas the outsourcing literature has established a two-domain taxonomy of business and technical knowledge, software maintenance research has emphasized application knowledge. Our results suggest that the two-domain taxonomy is likely to produce incomplete pictures of knowledge in SMO. In six of
six cases, application knowledge was more salient in the data than business or technical knowledge. Application experience was also more strongly related to cognitive load than were technical or business experience. Moreover, the participants largely perceived application knowledge as the most problematic knowledge domain.

These results have important implications for outsourcing research. They contribute to a better understanding of what knowledge is involved in SMO. It appears crucial to distinguish application knowledge from business and technical knowledge. Application knowledge is different because it is largely client-specific, but germane to the vendor’s task. Its initial levels and acquisition rates are therefore likely to be different from those of business and technical knowledge. Application knowledge is neither the sum of business and technical knowledge nor the ability to put technical knowledge into action. It is knowledge of how a given software reflects the many situated, path-dependent decisions that have been made over the life of a software application to match technology affordances with business needs. The following extreme case may help illustrate the difference. A VE be relocated from software A at client A to software B for client B. Assume that software A and B support highly similar business processes and use the same technology, with which the VE is familiar. Yet, social processes in the client organizations have produced different initial software designs and different modifications to the software over years of maintenance. As a consequence, software A be very different from software B. The VE may then struggle to comprehend the countless lines of code and the data in software B and to anticipate the consequences of maintenance actions, although the VE’s business and technical knowledge will certainly be useful. The three knowledge domains are thus not unrelated, but ignoring one may prohibit explaining outcomes such as cognitive overload in the example.

Studies of outsourced software projects should therefore adopt the three-domain taxonomy unless they refer to new-software development only. They should include application knowledge into statistical models because it may be the most important knowledge and because knowledge is strongly related to performance and other concepts prominent in outsourcing research such as organizational controls and trust [44]. Models without application knowledge may suffer from endogeneity and allow limited inference. This applies not only to SMO, but also to other software tasks that involve the client’s existing applications such as in outsourced software reengineering [1, 21]. Future work should distinguish software tasks in which engineers need to acquire application expertise from tasks in which they do not. Studies might therefore develop or test theories separately for software maintenance, reengineering, and new-software development.

The critical role of application knowledge has implications for how knowledge is exchanged in SMO. The two-domain taxonomy risks reducing knowledge processes to a knowledge integration problem in which clients and vendors coordinate their specialized business and technology expertise, while they do not necessarily acquire the other party’s specialized expertise. Studies that endorse this view see clients and vendors as “different expert groups” [35, p. 607]. However, this view is difficult to reconcile with our data. Although the VEs were experts in the technologies, they lacked knowledge of the software application, in particular in the cases of custom-developed software. If VEs initially lack application knowledge and if this knowledge is at the core of their task, then VEs are more novices than experts. They have limited domain-specific experience, suffer from cognitive overload, and depend on help. If VEs are novices, then knowledge processes in SMO are not only knowledge integration, but also, if not primarily, expertise acquisition. The difference is of high practical importance. SMO projects may use sophisticated knowledge integration tools such as direction and boundary objects [28], but these tools will not enable maintenance performance where prior maintainers have been dismissed and the remaining amount of application expertise to be integrated is low on both sides. It appears thus that SMO projects cannot solely rely on economically efficient knowledge integration, but that they need significant investments into the acquisition of application expertise by VEs. Projects that do not anticipate this need are likely to face severe extra costs [1] for helping cognitively overloaded VEs.

Some qualifications on the implications of our study are in order and may inspire future research. First, our study does not reject that the two-domain taxonomy is a useful taxonomy in the context of new-software development, the context in which it was proposed [16]. Second, our results do not imply that business knowledge acquisition is generally unimportant or less important than application knowledge acquisition in SMO. Five of the six cases involved requirements engineers to substitute for business knowledge transfer. Business knowledge might have played a more prominent role if requirements engineers had not been involved. While it may thus not be appropriate to generalize the relative importance of knowledge domains to other cases, our results do imply that ignoring application knowledge and its acquisition can produce erroneous theoretical inference. Finally, our study does not imply that knowledge integration is unimportant in new-software development or in software maintenance. In fact, we also observed knowledge integration practices. Instead, we submit that knowledge processes in
SMO cannot be reduced to knowledge integration. Knowledge processes may include both knowledge integration and expertise acquisition, and the relative emphasis might differ between projects according to application expertise and other factors. Future research could delve into this exciting issue. The key role of application expertise acquisition also invites future research about how VEs can acquire such expertise most effectively, opening the black box of the VEs’ learning processes.

Our study is not without limitations. While two coders have been involved at different stages of the project, knowledge domains were coded by the first author only. The use of three analysis strategies may partially compensate for this weakness. Moreover, we investigated six cases, all of which were at the same Swiss bank. Future research can replicate or extend the study in other industries and software types.

Although the paper primarily aims to stimulate future research, some practical implications for SMO may be sketched. Clients and vendors should plan for possibly significant investments into application expertise acquisition in their business cases. In doing so, they should consider the specificity not only of business processes, but also of software applications. They may consider software packages as preferred candidates for SMO. Given the critical role of application expertise and high turn-over rates at major vendors [1], client and vendors should also implement controls to retain VEs in the projects once they have acquired expertise, in particular in custom-developed software.

Acknowledgements

This work has been supported by the Swiss National Science Foundation (Grant # 100018_140407 / 1).

References

Projects - a Case from the Financial Services Industry", The 43th Hawaii International Conference on Systems Sciences, 2009


Appendix A: Coding Examples

<table>
<thead>
<tr>
<th>Application knowledge</th>
<th>knowledge about the software application, its structure, functionality, and behavior [19, 20]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subdomain framework:</td>
<td>&quot;He has to add an attribute to a particular table. Which files should be changed for that, that's typically application-specific. So there's a typical DDT XML file. And for adding that, you cannot write an Alter Table command. We have a framework that I told you. He has to add that piece of information in an XML file.”</td>
</tr>
<tr>
<td>Subdomains [component A] and [component B]:</td>
<td>&quot;[component A] sends the data to [component B]. So we have this data in [component B].&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Business knowledge</th>
<th>knowledge about the client’s business processes, rules, activities, and objectives [16]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subdomain object type hierarchy:</td>
<td>&quot;We have four levels. 1 is the account, 3 is the customer, 4 is the relationship manager, and 5 is the organizational unit.”</td>
</tr>
<tr>
<td>Subdomain case workflow:</td>
<td>&quot;The application is related to work flow. So when a party is changing or when a case is changing hand, how much time it takes, who is closing the case.”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Technical knowledge</th>
<th>knowledge about design, programming, and software processes [16]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subdomain deployment:</td>
<td>&quot;The procedure for the deployment is that we work on the development environment, then it's loaded into the test environment and then it goes to production.”</td>
</tr>
<tr>
<td>Subdomain PL-SQL:</td>
<td>&quot;Yes, this was a PL-SQL procedure. My tasks are entirely PL-SQL.”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cognitive load:</th>
<th>The mental demands that a task imposes on a learner; coded on a 5-points scale based on perceived difficulty, mental effort, and task performance [41]</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 out of 5 (task not feasible):</td>
<td>&quot;He did not know how he could test that.” (SME)</td>
</tr>
<tr>
<td>4 out of 5 (task solved with problems):</td>
<td>&quot;The instruction [that I had given to the other team in the deployment procedures] was not proper” (VE)</td>
</tr>
<tr>
<td>3 out of 5 (task difficult, but solved):</td>
<td>&quot;It was demanding” (VE), the change request was considered successful by the client</td>
</tr>
<tr>
<td>2 out of 5 (task moderately difficult and solved):</td>
<td>&quot;It is a little bit complex only” (VE), the change request was considered successful by the client</td>
</tr>
<tr>
<td>1 out of 5 (task easy and solved):</td>
<td>“The task was clear cut.” (VE)</td>
</tr>
</tbody>
</table>