

# Beyond the Black Box: Knowledge Overlaps in Software Outsourcing

**Amrit Tiwana**, *Emory University*

**T**he black-box approach (that is, the use of formal project requirements to transfer knowledge about the application problem domain from the client to the vendor organization) has long been the mainstay of outsourced software development. Both client and vendor organizations benefit from this approach because they rarely need in-depth knowledge about the others' domain. For some projects, however, effective outsourced software development requires organizations

to break out of the black-box mold. What types of projects require the vendor to possess in-depth knowledge of the client's business or the client to have in-depth technical knowledge? What deviations from the black-box approach are necessary when the project is conceptually innovative or involves novel software development processes?

To shed light on this issue, at Emory University, we've studied 209 custom application development projects in 209 global software development organizations. All outsourcing organizations were American corporations, and the vendors to whom they outsourced

their projects were key member software development organizations in three of the largest global software development consortia in India, Ireland, and Russia.

We used the study results to develop detailed guidelines for improving outsourcing practice and a client-vendor knowledge congruence assessment framework (see the "Outsource or Insource?" sidebar). The study's key finding is that effective outsourcing requires knowledge congruence—that is, a good fit in terms of the business and technical knowledge across the client-vendor dyad. This uniquely extends the notion of the product/process fit from the manufacturing management arena into the more knowledge-driven domain of software engineering.<sup>1,2</sup>

## Software as embodied expertise

Good software is an accurate embodiment

Although the black-box approach to software development outsourcing is effective for routine projects, projects that use novel concepts or processes require some knowledge overlap in client and vendor organizations.

# Outsource or Insource? A Client-Vendor Knowledge Partnership Congruence Framework

To evaluate knowledge congruence, or fit between the nature of the project and the vendor to which the project might be outsourced, client organizations must first assess the extent to which a project is conceptually innovative and involves novel development processes or tools. Based on this assessment, clients must seek to closely match their choice of vendor to correspond to the appropriate knowledge overlap pattern (see Figure 4 in the main article). Our knowledge congruence framework consists of four steps for identifying this match.

The figures accompanying each step are worksheets. Replace the sample scores in the worksheets with your own project's scores.

## Step 1: Determine project-specific novelties

The first step is to determine the type of novelties associated with the project. To assess conceptual novelty, consider the vendor's prior development experience, references, and your understanding of the vendor's expertise. Assign scores between 1 and 4 for the level of novelty in the project's underly-

Assessment area	Score
Design concept	3
System design	4
System functionality	3
Business application problem domain	4
Conceptual novelty score (add rows)	14

1 = minor modification of a pre-existing system,  
 2 = major modification of a pre-existing system,  
 3 = completely new design based on a proven concept  
 4 = unprecedented design

Figure A. Assessing a project's concept novelty.

ing concepts in Figure A and add the individual scores for a total score. If the total score is between 4 and 10, conceptual novelty is low; between 11 and 16, novelty is high.

To assess process novelty, consider the software development methodologies and tools the vendor has used in the past. The scores in Figure B correspond to the level of novelty in the project's processes. A process novelty score between 2 and 5 indicates low process novelty; a score between 6 and 8 indicates high process novelty.

## Step 2: Assess your organization's internal technical expertise

The second step involves assessing your organization's internal technical expertise. Using a scale of 1 (very low) to 10 (very high and close to perfect), how would you rate the technical expertise of your organization's internal IS department in the various areas specific to this project, such as those in Figure C? Consider these questions from the hypothetical perspective of developing the system in-house. A score between 6 and 33 indicates that the client's internal

Assessment area	Score
Development methodology	2
Software development tools	4
Process novelty score (add rows)	6

1 = existing processes with minor modification,  
 2 = existing processes with major modification,  
 3 = new processes derived from existing ones  
 4 = entirely new and untried processes

Figure B. Assessing a project's process novelty.

of application-problem-domain expertise and technical expertise.<sup>3-5</sup> To effectively meet its client's business needs, the software's design must represent both knowledge types.<sup>6-9</sup> Two properties of such knowledge complicate its application to the development process:

- The knowledge is distributed<sup>10</sup> on either side of the client-vendor boundary. In an outsourced application development project, the client organization maintains knowledge about the project's business needs and the vendor organization maintains technical expertise to develop the software.
- Some of the knowledge is so complex and context dependent<sup>15</sup> that it's difficult to articulate and transfer across the client-vendor interface. Leveraging such distrib-

uted knowledge requires moving it from the organization in which it resides to the organization that will use it.

Fortunately, neither fragmentation nor context dependence poses a compelling problem in routine software projects. As Figure 1 illustrates, formal requirements help move knowledge about the project's business needs from the client to the vendor (the *business requirements loop*) and established, agreed-upon development processes create a feasible *design validation loop* for iteratively refining these requirements. This pattern usually works well. The vendor needs only some basic knowledge about the client's business, and the client needs limited technical expertise for this approach to suffice. With annual global spending on software outsourcing now exceeding US\$500 bil-

technical expertise is low; a score between 34 and 60 shows a high level of expertise.

### Step 3: Assess vendor expertise

Using a scale of 1 to 10 as in Step 2, how would you assess the vendor's understanding of various areas of your business associated with this particular project? Figure D lists business application domains to be assessed along with sample scores. A score between 6 and 33 indicates low business application domain knowledge; a score between 34 and 60 shows high domain knowledge.

### Step 4: Assess outsourcing viability

The final step is to assess whether outsourcing to this vendor is a viable option for this project.

Figure E uses scores from Figures A–D and the profiles from Figure 4 (a–d) in the main text to determine whether an

organization should outsource a particular software development project.

Assessment areas	Score
Business processes	3
Application problem domain	4
Day-to-day business routines	9
Business rules to be implemented in the system	10
Your business objectives for this project	1
Familiarity with existing systems in your organization with which this system must interoperate	4
Vendor's business application domain knowledge score (total)	31

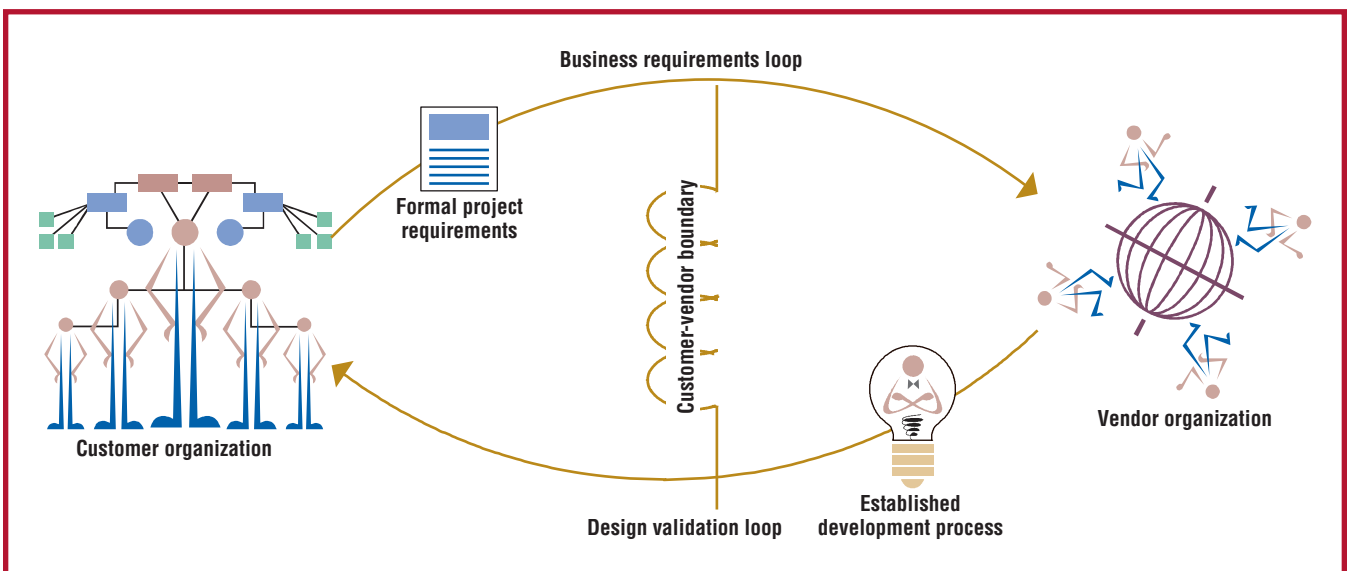
**Figure D. Samples scores in assessing vendor's business application domain knowledge.**

Assessment area	Score
Detailed systems design	10
Understanding of technical design constraints	6
Programming language to be used for this project	4
Understanding of various development methodologies	10
Testing and debugging procedures for software code	10
Familiarity with software development coordination tools (such as those used for requirements management, modeling, test case development, configuration management, and defect/change request tracking)	7
Client's technical knowledge score (add rows) ‡	47

**Figure C. Samples scores in assessing client organization's internal expertise.**

Assessment areas	Score
Business processes	3
Application problem domain	4
Day-to-day business routines	9
Business rules to be implemented in the system	10
Your business objectives for this project	1
Familiarity with existing systems in your organization with which this system must interoperate	4
Vendor's business application domain knowledge score (total)	31

**Figure E. Determining the feasibility of outsourcing a project according to project profile.**



**Figure 1. Boundary crossing using a black-box approach.**



**Process novelty  
in software  
development  
arises from the  
use of new  
software  
development  
methodologies,  
new  
development  
tools, or both.**

lion,<sup>11</sup> it's even more important to understand when the black-box approach doesn't work.

### **When little knowledge is a dangerous thing**

Associating novelty with a project highlights the weaknesses of the black-box approach. Novelty can exist in both a software development project's underlying concepts and in the development processes.

*Conceptual novelty* occurs when the project concepts and ideas are completely new to the vendor or the project attempts to solve a problem for which no precedent solution exists. *Process novelty* occurs when the project concepts are relatively well-understood but the vendor uses new development processes and tools. In either case, moving knowledge in one of the two loops is so difficult that the approach illustrated in Figure 1 isn't feasible. To move the knowledge across the client-vendor boundary, one organization must then possess more knowledge about the others' domain. Determining which one is the tricky part; a wrong assessment can wreck even a promising project.

### **Concept novelty**

The costliest and least remediable defects in software arise from misunderstood business requirements.<sup>12,13</sup> When the project concept is novel, it's difficult for the client to faithfully convey unique problems, novel ideas, and unfamiliar project concepts to the vendor through formal requirements. (Prior studies on hardware manufacturing refer to this type of novelty as *product concept novelty*.<sup>1</sup>) In the presence of conceptual novelty, the mechanisms that normally lubricate the business requirements loop underperform. Even when full cooperation exists at the client-vendor boundary, the vendor might inconsistently, ambiguously, and inaccurately interpret client needs, leading to design decisions that are incompatible with the client's project objective. Thus the problem is rooted in the vendor's inability to understand a conceptually novel project's needs. In such projects, the vendor needs a higher level of knowledge about the client's business application domain. In our study, we found that conceptually novel projects meeting this criterion resulted in significantly better software design and also deviated less from their planned budget.

### **Novel development processes**

Process novelty in software development arises from the use of new software development methodologies, new development tools, or both. Novel processes offer no established, mutually understood syntax through which the client can monitor and provide feedback to the vendor during development. Thus the mechanisms that normally create the design validation loop are infeasible. Even full cooperation between the two organizations can't fully compensate for the client's inability to understand the development process. After all, half of the reasons for the black-box approach's effectiveness involve the client's ability to differentiate effective from ineffective development practices and to understand and validate artifacts such as specifications, prototypes, and mock-ups generated during the development lifecycle. To understand and act on vendor-generated information during development of a project involving novel development processes, the client organization needs to better understand the project's technical intricacies.

### **International field study**

To determine when and why certain knowledge overlap patterns are necessary in outsourced software development projects, we conducted a large-scale international field study of vendors and their clients. The study consisted of three phases.

In the first phase of the study, we interviewed 19 project managers from 19 organizations and seven academic software development experts. The 19 project managers represented the entire spectrum of organizations in the second phase of the study; six were project managers from American firms that contracted development to outside software development organizations. To avoid bias, we included none of the organizations from the first phase of the study in the later phases. We used data from the interviews to develop a questionnaire for the subsequent study phases (the "Survey Measures Overview" sidebar describes these measures).

In the study's second phase, we contacted 818 software development organizations through the three largest global software consortia in Russia, Ireland, and India: the Russian National Software Development Alliance, Irish Investment and Development Agency, and India's National Association of Software

and Service Companies. We initially asked the president or CEO of each vendor company to identify a recent project they had completed for a US client. We then asked the project's lead project manager to complete the questionnaire. This yielded vendor-side project data on 232 projects in 232 organizations—an overall response rate of 28.4 percent, with responses from 59 Russian, 54 Irish, and 119 Indian software development organizations. We also collected data on the defects recorded at each stage of the development process and the percentage by which each project exceeded the original schedule and budget.

In the third phase, we collected US client evaluations of the same projects. Eventually, we collected project-level data from two sets of key informants from each organization—project managers in the vendor organization and their client-side liaison managers. We obtained client evaluations of all but 23 projects, resulting in a final matched-pair sample of 209 projects.

### Project characteristics

Figure 2 shows the distribution of process and conceptual novelty for the projects in the study. All projects in the study were custom application development projects, and only 10 percent were truly routine, involving no novelty. Sixty-eight percent of the projects involved conceptual novelty: approximately 40 percent were based on a preexisting design and 28 percent were truly conceptually novel. Only 9 percent of the projects used entirely

## Survey Measures Overview

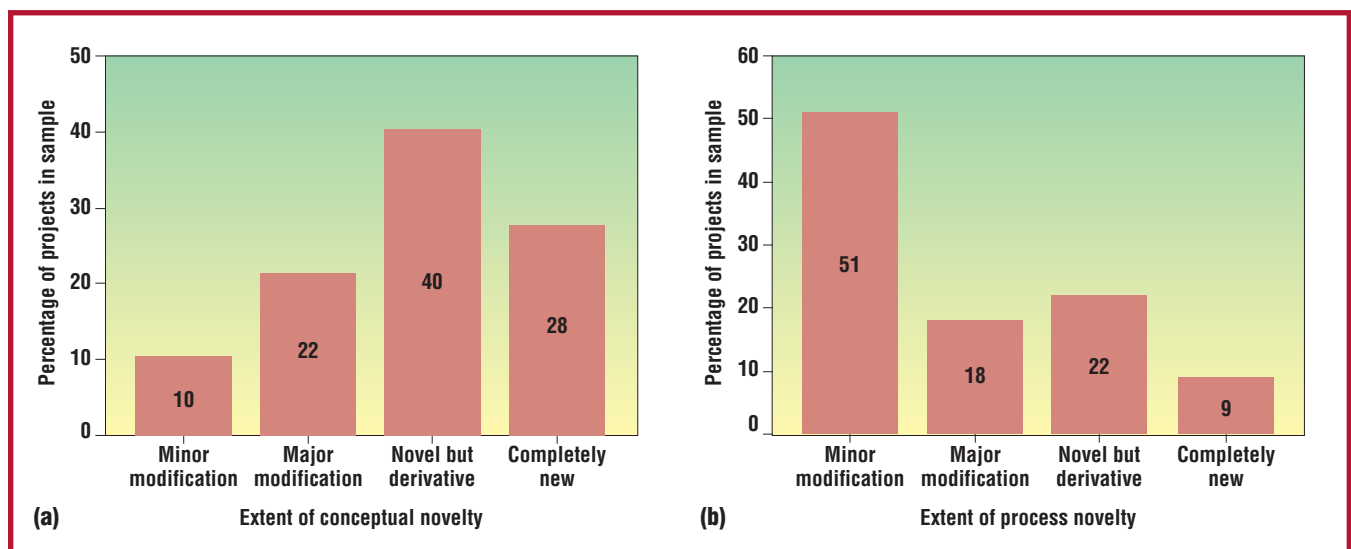
As part of an international field study of client and vendor organizations involved in software development outsourcing, 232 project managers from software development firms completed questionnaires. Various empirical measures evaluated knowledge levels and knowledge overlaps. All measures used multiple questionnaire items:

- *Client's technical knowledge* measured the client organization's technical knowledge about project-specific tools, processes, development, and coding.
- *Vendor's business knowledge* measured the vendor organization's knowledge about the client's business application domain.
- *Knowledge integration* measured how coherently the organizations combined application domain knowledge and technical knowledge in formulating project concepts, design, and solutions.
- *Conceptual novelty* measured the degree to which the software application was based on an unproven concept and had no technical precedent.
- *Process novelty* measured the newness of the software development methodology and development.
- *Client-vendor collaboration* measured the collaboration between the client and vendor organization over the project's development life cycle.
- *Development coordination tool usage* measured the extent to which vendors used tools for requirements management, modeling, test case development and automation, configuration management, and defect/change request tracking in the development process.
- *Architectural design effort* measured the percentage of project hours the vendor spent on the system's upfront design phase vis-à-vis project management, coding, and testing.

novel development processes; 51 percent involved no process novelty.

Project duration ranged from one month to

**Figure 2. Distribution of projects in terms of (a) conceptual novelty and (b) process novelty.**



Knowledge type	Project type		
	Routine	Conceptually novel	Novel processes
Client organization's technical knowledge	-	No statistical evidence found	+
Vendor organization's business application domain knowledge	No statistical evidence found	+	+
Corresponding optimal client-vendor knowledge overlap pattern			

**Figure 3. Results from the international field study of 209 outsourced projects.**

four years, with an average duration of about 11 months. The average size of the team dedicated to the project was 16 people (the number varied from two to 150). Of the vendors that had worked with the same client on earlier projects (about 59 percent), the average relationship spanned 3.6 years. On average, 20.7 percent of the total project hours (standard deviation 8.47) were spent on the project's upfront design. The vendors' capability maturity model (CMM) levels varied from 0 to 5 (the mean was 2.6). On average, the vendors in the study had been in the software development business for about seven years (from startup organizations to a 47-year veteran). Knowledge patterns varied widely across companies in the study.

In terms of project outcomes, on average, the projects exceeded the original schedule by 18.95 percent (standard deviation 19.7 percent) and the original budget by 15.7 percent (standard deviation 16 percent). On a scale of one to seven, the clients rated the projects' overall success an average of 5.4 (standard deviation 0.91).

### Summary of analyses

We conducted the analysis using a multiple regression model, denoting statistical significance by a 95-percent confidence level that the results were not by chance and drawing conclusions only from statistically significant relationships. (Detailed statistical analyses are available elsewhere.<sup>9</sup>) Figure 3 summarizes the analyses' key results (shaded cells indicate statistical significance).

A positive sign in the shaded cells indicates that the given knowledge type improved project outcomes; a negative sign indicates that it worsened outcomes.

The last row shows the corresponding knowledge overlap pattern for each project type based on these results. As the figure shows, the black-box approach is optimal in routine projects. However, in conceptually novel projects, the vendor must possess higher levels of knowledge about the application problem domain; in projects involving novel software development processes, the client must possess higher levels of technical knowledge.

### Key findings

Figure 4 summarizes the knowledge overlaps across client and vendor organizations necessary for effective outsourced software development in the presence and absence of novelty. Here, business knowledge refers to knowledge about the client's business problem application domain. When overlap patterns match the type of novelty characterizing a project, the project is more likely to be effectively and efficiently completed. Effectiveness here refers to the extent to which the project solves the client organization's problem (the client organizations provided the effectiveness assessments). Efficiency refers to the extent to which the project was completed within budget.

Overlaps that are consistent with the patterns in Figure 4 enhance the design process quality. Improved design processes increase software development efficiency by reducing defect rates throughout the development tra-

jectory and eventual rework-related cost overruns. Similarly, they improve design effectiveness by achieving a closer fit between the delivered system and client needs.

## Outsourcing guidelines

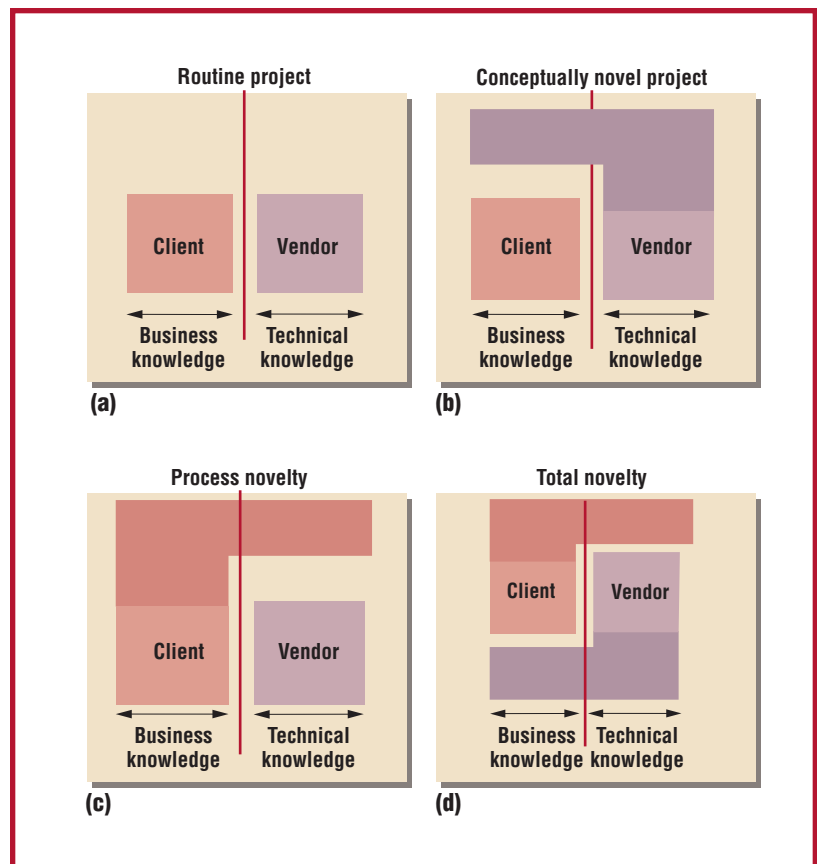
Do the study results show that software development organizations should invest in increasing their developers' business knowledge, and organizations that outsource development should increase their employees' technical expertise? The answer is both yes and no.

A vendor that regularly develops routine applications doesn't directly benefit from acquiring additional expertise in prospective clients' business domains. In such cases, a high level of technical expertise in client organizations actually reduces the design effectiveness for the project.

In projects involving process novelty, a high level of business knowledge in the vendor organization negatively affects the development process (therein lies the underappreciated danger of knowing too much). This finding could indicate that vendors with very high levels of business knowledge can sometimes be blind to the intricacies of the client's problem and thus make assumptions that impede the integration of contextual knowledge during the development process.

The key take-away: blindly acquiring knowledge outside of the software development domain is as perilous for vendors as is technically educating their clients. Managers considering outsourcing should therefore form carefully pruned outsourcing partnerships consistent with the patterns in Figure 4.

When an outsourcing relationship matches the profile in Figure 4d (that is, total novelty), the single most effective compensating mechanism is close vendor-client interaction throughout the development process. What other development coordination tools and mechanisms can help? We evaluated six popular development coordination tools (requirements managers, architectural modelers, test-automation tools, test-case development tools, configuration managers, and defect and change request-tracking tools), vendor CMM<sup>12</sup> level, and larger investments in the preliminary architecture design phase. However, we found no evidence that any of these improved design quality in the presence of total novelty as did ongoing interaction between



**Figure 4. Knowledge overlaps in outsourced software development: (a) Routine projects require no overlaps; (b) conceptually novel projects require vendors to have higher business domain expertise; (c) projects involving novel development processes demand higher technical expertise in the client organization; and (d) totally novel projects require extensive client-vendor communication across all phases of the development process.**

client and vendor. This is consistent with prior studies that have emphasized the importance of communication among stakeholders in the software development process.<sup>14</sup>

Given the widely held but anecdotal beliefs about software development coordination tools, it's not surprising that the vendors in the study used such tools extensively. The conditions that predict the effectiveness of widely used tools clearly need further research attention. Postwaterfall development methodologies such as agile development and pair-programming are noteworthy because they go to the heart of what was most helpful in the 209 projects we studied—the intensity and nature of interactions at the client-vendor boundary. However, organizations shouldn't rush to adopt such methodologies unless they have prior experience with them because hasty adoption of new methodologies itself might introduce process novelties. These methodologies also require more extensive (than the traditional waterfall methods) and ongoing interactions between the client and vendor organizations, which might be infeasible in an offshore outsourcing relationship.


## About the Author

**Amrit Tiwana** is an assistant professor in Emory University's Goizueta Business School in Atlanta. His research focuses on knowledge management in software development. He earned his PhD in management information systems from Georgia State University's Robinson College of Business. Contact him at Emory Univ., Goizueta Business School, 1300 Clifton Rd., Atlanta, GA 30322; [atiwana@bus.emory.edu](mailto:atiwana@bus.emory.edu).

**J**ust as no amount of engine power can make a poorly designed airplane fly, no amount of project management ingenuity can compensate for a poorly designed outsourcing partnership. Choosing a vendor that fits the appropriate pattern is easier than changing knowledge overlap patterns to fit a vendor.

Although the findings presented here offer no silver bullet, they offer a simple yet powerful approach for designing effective outsourcing relationships and for deciding when to avoid outsourcing altogether. The "Outsource or In-source?" sidebar presents a client-vendor partnership congruence framework based on these

findings. Collectively, the findings provide a powerful framework for evaluating what constitutes a good fit in outsourcing partnerships.

Managers considering outsourcing must evaluate their choices about the potential for outsourcing development as well as the choice of vendor on a project-by-project basis. Although a vendor's knowledge of a given organization's business practices and the industry are difficult to assess objectively, the vendor's past projects can provide some insight. Avoided redesign and rework can repay the investment in forethought about the black-box model's project-specific boundaries several times over. 

## References

1. P.S. Adler, "Interdepartmental Interdependence and Coordination—the Case of the Design/Manufacturing Interface," *Organization Science*, vol. 6, no. 2, 1995, pp. 147–167.
2. D. Nadler and M. Tushman, "Organizational Assessment," *Competing by Design*, Harvard Univ. Press, 1980, pp. 261–278.
3. P. Armour, "A Case for a New Business Model: Is Software a Product or a Medium?" *Comm. ACM*, vol. 43, no. 8, 2000, pp. 19–22.
4. P. Robillard, "The Role of Knowledge in Software Development," *Comm. ACM*, vol. 42, no. 1, 1999, pp. 87–92.
5. I. Rus and M. Lindvall, "Knowledge Management in Software Engineering," *IEEE Software*, vol. 19, no. 3, 2002, pp. 26–38.
6. B. Adelson and E. Soloway, "The Role of Domain Experience in Software Design," *IEEE Trans. Software Eng.*, vol. 11, no. 11, 1985, pp. 1351–1360.
7. S. Faraj and L. Sproull, "Coordinating Expertise in Software Development Teams," *Management Science*, vol. 46, no. 12, 2000, pp. 1554–1568.
8. V.S. Mookerjee and R. Chiang, "A Dynamic Coordination Policy for Software System Construction," *IEEE Trans. Software Eng.*, vol. 28, no. 6, 2002, pp. 684–694.
9. A. Tiwana, "Knowledge Partitioning in Outsourced Software Development: A Field Study," *Proc. Int'l Conf. Information Systems*, Assoc. for Information Systems, 2003, pp. 259–270.
10. B. Ramesh, "Process Knowledge Management with Traceability," *IEEE Software*, vol. 19, no. 3, 2002, pp. 50–55.
11. M. Lacity, "Lessons in Global Information Technology Outsourcing," *Computer*, Aug. 2002, pp. 26–33.
12. W. Gibbs, "Software's Chronic Crisis," *Scientific Am.*, Sept. 1994, pp. 86–95.
13. R. Rowen, "Software Project Management under Incomplete and Ambiguous Specifications," *IEEE Trans. Eng. Management*, vol. 37, no. 1, 1990, pp. 10–21.
14. C. Seaman and V. Basili, "Communication and Organization: An Empirical Study of Discussion in Inspection Meetings," *IEEE Trans. Software Eng.*, vol. 24, no. 7, 1998, pp. 559–572.
15. E. von Hippel, "Sticky Information and the Locus of Problem Solving: Implications for Innovation," *Management Science*, vol. 40, no. 4, 1994, pp. 429–439.



**Become an  
IEEE  
Software  
reviewer**

The key to providing you quality information you can trust is *IEEE Software's* peer review process. Each article we publish must meet the technical and editorial standards of industry professionals like you. Volunteer as a reviewer and become part of the process.

**Become an *IEEE Software* reviewer today!**  
Find out how at [www.computer.org/software](http://www.computer.org/software).

For more information on this or any other computing topic, please visit our Digital Library at [www.computer.org/publications/dlib](http://www.computer.org/publications/dlib)