# Contextualized Relationship Between Knowledge Sharing and Performance in Software Development

MUAMMER OZER AND DOUG VOGEL

MUAMMER OZER is a professor of management and director of the Doctor of Business Administration program at the City University of Hong Kong. He received his Ph.D. in business administration from the University of Pittsburgh. His current research interests include the software development process, innovation management, and knowledge management. His work has appeared in journals such as *Journal of Applied Psychology* and *Strategic Management Journal*.

DOUG VOGEL is chief academic director of Management Science and Engineering at Harbin Institute of Technology in China. Previously, he was chair professor of information systems at the City University of Hong Kong. He received his Ph.D. in information systems from the University of Minnesota. He is a fellow and former president of the Association for Information Systems. His current research interests include mobile devices, virtual world support, and educational systems. His work has appeared in journals as such *Journal of Management Information Systems* and *MIS Quarterly.*

ABSTRACT: We study how the knowledge that software developers receive from other software developers in their company impacts their performance. We also study the boundary conditions of this relationship. The results of our empirical study indicate that receiving knowledge from other software developers in the company is positively related to the performance of the knowledge-receiving software developers. Moreover, this relationship was stronger when the software developers had high rather than low task autonomy, when they had high- rather than low-quality social exchanges with their supervisors, and when the software development firms used formal knowledge utilization processes. Theoretically, these results contribute to a better understanding of the processes through which software developers utilize the knowledge that they receive from their peers in the firm. Practically, they show software development firms how emphasizing the task, social, and institutional dimensions of the software development process can help them increase knowledge utilization and performance in software development.

Muammer Ozer acknowledges generous financial support from the Department of Management at the City University of Hong Kong.

*Journal of Management Information Systems* / 2015, Vol. 32, No. 2, pp. 134–161.

ISSN 0742–1222 (print) / ISSN 1557–928X (online)
DOI: 10.1080/07421222.2015.1063287

KEY WORDS AND PHRASES: knowledge sharing, software developers, software development, software-development performance, systems design and implementation.

Software products and services enjoy a very sizable market, generating annual revenues of more than $300 billion in the United States alone and significantly contributing to economic growth and employment [52]. At the same time, the challenge to develop successful software projects could not be higher, as a large proportion of software projects are historically categorized as either incomplete or challenged, and are overbudget, late, or completed with fewer features and functions than originally envisioned [42].

Given such high stakes in software development, it is not surprising to witness a wide range of suggestions in the literature that can help improve the quality and productivity of the software development process, including follow-the-sun workflows [6], eXtreme Programming [16], computer-aided software engineering (CASE) tools [22], team cognition [23], service-oriented methodologies [27], goal setting [31], project coordination [32], novelty-knowledge alignment [50], agile systems [57], and software process tailoring [60], among many others.

Although these technology- and/or process-oriented approaches to improve the software development process are important because software development is a knowledge-intensive activity that entails software developers to utilize knowledge [12, 44], it is also important to study how knowledge sharing impacts performance in software development, especially given that the shared knowledge needs to be integrated into one's existing knowledge and applied in problem solving [51]. In fact, while the creation, storage/retrieval, and transfer of knowledge are, in general, considered to be important in knowledge management, they do not necessarily lead to enhanced performance unless the knowledge is actually applied in the organization [1]. However, despite the advances in knowledge sharing and application methodologies, shared knowledge is not always applied in software development [9].

In this study, we build our theory on the application aspect of shared knowledge within the context of software development and focus primarily on how the knowledge that software developers receive from other software developers in the firm might impact their performance. We draw on task closure theory [46] to study this relationship. Originally introduced to explain how selecting media that bring closure to a communication task can enhance communication among knowledge workers, this theory, in essence, emphasizes the need to bring closure to a communication activity. A knowledge worker shares his/her knowledge with another knowledge worker with the expectation that the latter will act on that knowledge [48]. Otherwise, an incomplete communication can lead to fragmentation of work, to stress, and subsequently to lower performance [46]. In a software development context, application of the shared knowledge in one's software development activities brings closure to the knowledge sharing task.

A number of factors can lead to closure in knowledge sharing. For example, a conceptual study highlighted the important role of information technologies that can facilitate the wider capture and accessibility of knowledge in organizations [1]. In addition, it was shown that the motivational considerations of knowledge providers and recipients impacted the way they transferred and used knowledge in a simulated environment [40]. Finally, in an experimental study, it was found that knowledge demonstrability and the similarity of the social identities of the knowledge providers and recipients facilitated knowledge transfer and usage [25].

In this study, we integrate task closure theory with three complementary theories related to knowledge application contexts to study how software developers apply the knowledge that they receive from other software developers in their firms and thus bring closure in their knowledge sharing. Both the academic and practical importance of knowledge application contexts has long been recognized in the literature [1]. In fact, it was noted that hindrance to knowledge transfer and application (i.e., task closure) in knowledge sharing largely stems from the circumstances surrounding the knowledge recipients [25]. Software development takes place in a multidimensional context, while software developers need to focus on their software development tasks (i.e., task context), they need to constantly interact with their supervisors (i.e., social context), and perform their tasks within a larger institutional setting with both implicit and explicit rules, standards, documents, and metrics (i.e., institutional context). Because these different aspects of the software development context are likely to impact knowledge utilization in software development (i.e., task closure) differently, we take a finer-grained approach to study the relationship between received knowledge and performance as moderated by these multi-dimensions of the software development context. This is also consistent with the growing recognition in the information systems (IS) literature of the need to use a multi-domain view in IS research [15]. We summarize our conceptual model in Figure 1 and explain it below.

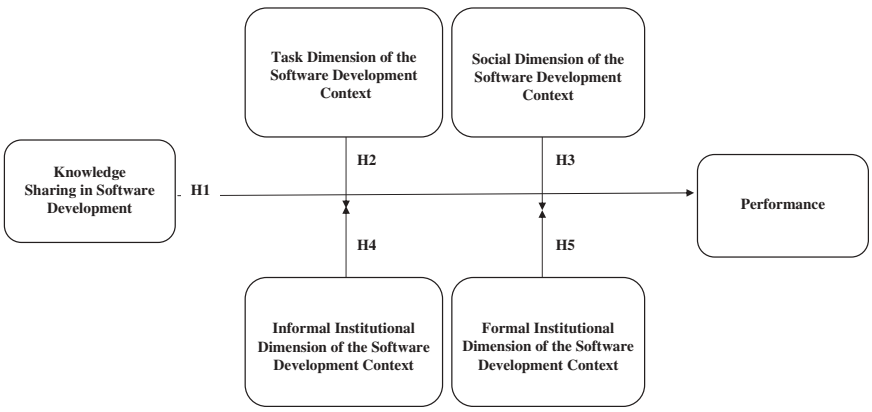


*Figure 1.* Conceptual Model

First, software developers need to design software features and detect as well as correct glitches, coding errors, and other software flaws [54]. However, the way software developers conduct these activities and the way they utilize the knowledge that they receive from other software developers in the firm depends primarily on how their job is designed. Consequently, we integrate task closure theory with the job design model [21] to study the relationship between received knowledge and performance in software development. One of the important aspects of the job design model is that job characteristics determine the level of discretion and control that an employee might have to carry out his/her job tasks [21, 26]. As we will discuss in more detail in the "Theoretical Development" section below, software developers' ability to bring closure to knowledge sharing in their software development activities will likely depend on the amount of freedom and discretion that they have to apply the knowledge that they receive from other software developers in the firm in their software development activities (i.e., task autonomy) [30]. Therefore, we test the moderating role of task autonomy in the relationship between received knowledge and performance in software development.

Second, we integrate task closure theory with social exchange theory to study how software developers can bring closure to knowledge sharing and apply the knowledge that they receive from other software developers in the firm into their software development activities. The important role of social context has long been recognize in IS research [7, 26]. In particular, it has been suggested that "people skills" in project leadership play an important role in successful software development [42]. It has long been known that when people engage in high-quality social exchanges with their supervisors, also known as leader–member exchange (LMX) [19], they will likely receive additional high-quality knowledge from their supervisors. Thus, it is likely that the LMX relationship of software developers with their supervisors (i.e., social context) will moderate the relationship between received knowledge and performance in software development. Hence, we also study the moderating role of LMX in this relationship.

Finally, software development takes place in an institutional setting with both implicit and explicit rules, standards, documents, and metrics [45, 49]. Thus, we integrate task closure theory with the information integration model [20] to study how software developers can bring closure to knowledge sharing and apply the knowledge that they receive from other software developers in the firm in their software development activities. Grant [20] conceptualized that knowledge in a firm can be integrated into one's work through informal organizational routines and/or explicit rules and directions. Within the software development context, while informal knowledge utilization processes tend to involve urging software developers to seek and utilize knowledge in software development in general, formal knowledge utilization processes tend to involve the use of more explicit rules, standards, documents, and metrics that software developers should apply in software development [45, 49]. Because these two types of institutional processes will likely play different facilitating roles in knowledge sharing and utilization, we study how these

two types of processes might moderate the relationship between received knowledge and performance in software development.

In summary, this research contributes to research on software process improvement by focusing on human- rather than technology- and/or process-oriented aspects of the software development process. In addition, it contributes to research on knowledge sharing and utilization by testing factors that influence knowledge utilization in software development. Finally, it integrates task closure theory with three complementary theories related to the different dimensions of the software development context to study how software developers apply the knowledge that they receive from other software developers in their firms and thus bring closure in their knowledge sharing. Consequently, this integrated and fine-grained framework allows for a more specific articulation of the processes through which software developers utilize the knowledge that they receive from other software developers in the firm.

## Theoretical Development

*The relationship between received knowledge and performance*: Knowledge-based theory posits that firms can achieve greater competitive advantages and increase their performance through their abilities to generate, store, transfer, and apply knowledge [1, 20]. Individuals are considered an important block in knowledge-driven value creation such as software development [8, 38]. As an important knowledge application domain, software development has long been recognized as a complex problem-solving process with short time horizons [12, 44]. And because most software development problems are ill-defined, it requires defining relevant dimensions of the software development process, crafting a feasible and acceptable solution, and convincing all the relevant parties of the appropriateness of the proposed course of action [12].

Task closure theory [46] emphasizes the need to bring closure to a communication task in order for the communication between two knowledge workers to be effective. In a software development context, the knowledge sharing task will not be considered complete unless the shared knowledge is applied in one's software development activities. When bringing closure to the knowledge sharing process, software developers will likely use the knowledge that they receive from other software developers in the firm to guide them as to how to perform their software development tasks [51]. For example, software developers can learn from other developers in the firm how to code specific software functions and features, how to detect software flaws, and how to fix them in a timely fashion. In addition, software developers can learn from other developers in the firm how to define relevant dimensions of the software development process, craft more feasible and appropriate solutions, and convince all the stakeholders of the software development process of the suitability of the proposed solutions based on that knowledge and hence perform better [12]. As a consequence, we posit that:

*Hypothesis 1: Receiving knowledge from other software developers in the firm will have a positive impact on the job performance of the knowledge-receiving software developers.*

*The moderating role of the task dimension of the software development context in the relationship between received knowledge and performance*: Software developers need to design software features and detect as well as correct glitches, coding errors, and other software flaws [54]. However, the way software developers conduct these activities and the way they utilize the knowledge that they receive from other software developers in the firm primarily depends on how their job is designed. An important aspect of the job design model involves the level of discretion and control that an employee might have to carry out his/her job tasks, which is also known as task autonomy [21, 29]. Although task autonomy appears to be related to performance, it taps into a distinct theoretical and conceptual domain, while task autonomy is related to the job design model [21, 29] performance is explained by role theory [58]. In general, role theory suggests that an employee's performance reflects his/her responsibilities associated with his/her employment as well as how proficiently he/she can fulfill these responsibilities in order to attain higher-level organizational goals [58]. Past research has consistently found a large variation in the strength of the relationship between task autonomy and performance, even a negative relationship between the two [29]. This suggests that high task autonomy might not automatically translate into high performance, as task autonomy can be linked to performance through complex motivational, informational, and structural mechanisms that can enhance or hinder the strength of this relationship [29]. Similarly, a high performance might not automatically translate into task autonomy because task autonomy is usually part of broader job design and characteristics; because tasks will likely be interlinked with other tasks carried out by other individuals; and because some individuals, regardless of their performance, will likely desire to have task autonomy more than others [21, 29].

Software development is performed in constantly changing environments generally typified by unpredictable markets, changing software requirements, shortened delivery times, and rapidly advancing information technologies [53]. Hence, software developers need to have a considerable amount of discretion and freedom in performing their software development activities in order to respond to these daily demands more effectively and quickly [30]. As part of H1, we argued that the knowledge that software developers receive from other developers in the firm will likely guide them in performing their software development tasks [51] as well as in defining relevant dimensions of the software development process, crafting more feasible and appropriate solutions, and convincing all the stakeholders in the software development process of the suitability of the proposed solutions based on that knowledge [12]. Because task autonomy gives software developers the freedom and discretion to incorporate the knowledge that they receive from other software developers into their software development tasks [30] their performance can benefit from that knowledge if they are allowed to change their work based on that knowledge. On the other

hand, because low task autonomy does not give them the flexibility and freedom to change their work, software developers working under low task autonomy conditions will not able to use the knowledge that they receive from other software developers in the firm and thus will not be able to bring closure to their knowledge sharing activities with the other software developers in the firm. Consequently:

> *Hypothesis 2: Receiving knowledge from other software developers in the firm will have a stronger positive impact on the job performance of the knowledge-receiving software developers who have high rather than low task autonomy.*

*The moderating role of the social dimension of the software development context in the relationship between received knowledge and performance*: While software developers need to focus on their immediate software development tasks (i.e., task context), they also need to constantly interact with their supervisors (i.e., social context) in the software development context. LMX theory has been proposed as a major theory to study linkages between leadership processes and outcomes in organizations [19]. Unlike traditional leadership theories that focus on the leadership contexts or the personal characteristics of a leader or a follower, LMX theory emphasizes the social exchanges between a leader and his/her follower [17, 19]. As a result, it taps into dyadic characteristics of the relationship between a leader and his/her follower such as understanding, recognition, trust, respect, and mutual obligations [17, 19]. These theoretical characteristics differentiate LMX from performance, which is generally explained by role theory that highlights employees' responsibilities associated with their employment as well as how proficiently they can fulfill those responsibilities to attain higher level organizational goals [58]. In addition, meta-analytic studies report that the relationship between LMX and performance was modest ($r$ = .27) with a $Q$-statistic of 149.57. The large $Q$-statistic indicates that the correlation between LMX and performance varies substantially and the relationship between the two depends on numerous conditional factors. Indeed, a high-quality LMX relationship depends on the characteristics of the leader (expectations, transformational leadership, extraversion, etc.) and his/her follower (competence, agreeableness, conscientiousness, etc.) as well as the characteristics of the relationship (perceived similarity, affect/liking, assertiveness, etc.) rather than performance [14]. Similarly, a high-quality LMX relationship between a leader and his/her subordinate may not always translate into higher performance, as performance will primarily depend on the subordinate's capabilities and contingencies [58].

In a multicountry Delphi study about software project risk factors, software project managers indicated that one of the important project risk factors was the lack of "people skills" in project leadership [42]. In particular, it was suggested that managing schedules, technology, and project requirements but ignoring the people dimension of software development contributed to the failure of software projects [42]. Indeed, besides the project-related interactions between a software developer and his/her project leader there is likely to be a social exchange between them, as knowledge sharing and application will likely take place within a larger social context that includes software

project leaders [7, 26]. Within the context of software development, a high-quality LMX relationship between a software developer and his/her supervisor can help develop a shared understanding of the software development tasks, technical language, and, more importantly, what exactly the supervisor expects from the software developer [23]. Such a shared understanding will likely familiarize software developers with the software development process, including suitable software development procedures, steps, and actions, which, in turn, can lead to better coordination of the software development tasks and performance [23]. Furthermore, as software developers benefit from their high-quality social exchanges with their supervisors they will likely gain more experience and knowledge and will likely reciprocate these benefits by being more motivated to perform even the most mundane software development tasks [55]. All these benefits will enable them to bring closure to their knowledge sharing activities with other software developers in the firm. Consequently, we propose that:

> *Hypothesis 3: Receiving knowledge from other software developers in the firm will have a stronger positive impact on the job performance of the knowledge-receiving software developers who have high- rather than low-quality LMX relationships with their supervisors.*

*The moderating role of informal knowledge utilization processes in the relationship between received knowledge and performance*: Informal knowledge utilization processes generally involve undocumented but socially accepted norms and beliefs that software developers share about the importance and the value of knowledge sharing and utilization in software development [34, 45]. These kinds of processes encourage software developers to share knowledge with other software developers in the firm and leverage their expertise [18].

Compared to software developers working in software development firms without any informal knowledge sharing and utilization process, software developers working in software development firms with such processes will likely be more aware of the importance and the value of knowledge sharing and utilization and will thus more likely use in their software development activities the knowledge that they receive from other software developers in the firm, as this is a socially acceptable norm in the firm [18, 41]. For example, software developers can get together in an informal brown-bag lunch gathering and share their software development experiences. Because these kinds of informal practices are more social and people-oriented [41], software developers will know who might have the knowledge that they need in the firm and thus be able approach them with greater ease [18, 41]. In addition, the social and personal nature of these processes will allow software developers to ask questions about hard-to-understand tacit software development knowledge and hence further clarify its applicability in their software development tasks [18, 41]. Social and people-oriented norms will likely help software developers establish a shared syntax, and as a consequence, will enable them to share and understand software development knowledge with less physical and cognitive effort [18]. Because all of these benefits of informal knowledge utilization processes are

conducive to bringing closure to the knowledge sharing process and thus to better performance [18, 41], we expect that:

> *Hypothesis 4: Receiving knowledge from other software developers in the firm will have a stronger positive impact on the job performance of the knowledge-receiving software developers when software development firms use informal knowledge utilization processes in software development as opposed to when they do not.*

*The moderating role of formal knowledge utilization processes in the relationship between received knowledge and performance*: Formal knowledge utilization processes deal with the institutionalized knowledge sharing and utilization practices in software development [34, 45]. For example, software development firms can institutionalize knowledge sharing and utilization in software development and reward software developers based on their adherence to such processes [18]. In addition, they can create dedicated collaborative platforms for software developers to store and share knowledge pertaining to software development [4].

It is widely recognized that software development is already a highly unstructured and ambiguous problem-solving process [12, 44]. The use of formal knowledge sharing and utilization processes can help structure this process by providing software developers with clear objectives and a set of procedures for sharing and utilizing knowledge with each other [38]. Moreover, it leads them to focus their efforts on better utilization of the knowledge that they receive from other software developers in the firm [18]. Thus, compared to software developers working in a software development firm without any formal knowledge sharing and utilization process, software developers working in a software development firm with such processes will likely benefit more from the knowledge that they receive from other software developers in the firm and therefore bring closure to the knowledge sharing process with other software developers in the firm, which, in turn, will enable them to perform better [12]. It follows that:

> *Hypothesis 5: Receiving knowledge from other software developers in the firm will have a stronger positive impact on the job performance of the knowledge-receiving software developers when software development firms use formal knowledge utilization processes in software development as opposed to when they do not.*

# Methodology

## Procedure and Participants

We contacted software developers and their immediate supervisors participating in a large information technology (IT) exhibition in Asia in the spring of 2011. In order to minimize the common method bias [39], two separate questionnaires were designed, one for software developers and one for their immediate supervisors. Of the 568 firms participating in the exhibition, 320 indicated that they engaged in software

development. We distributed 946 paired questionnaires in those 320 firms. While software developers answered questions about receiving knowledge, task autonomy, LMX, and informal and formal knowledge utilization processes, their immediate supervisors answered questions about the job performance of the software developers based on the multi-item measures that we describe below. The questionnaires also covered a wide range of demographic variables. They included a cover letter summarizing the purpose of the study and assuring the respondents that the collected information would be used strictly for research purposes and would in no way reveal the respondents' identities. The questionnaire items are listed in the Appendix.

Overall, we collected 309 software developer and 177 supervisor questionnaires. The response rates were 32.7 percent and for the software developers and 55.3 percent for the supervisors. A total of 261 matched questionnaires (including supervisors' rating of the job performance of the software developers) were available for the analyses. Of the software developers, 72.8 percent were male. The mean age ranged from 26 to 50. All the software developers had at least a high school degree, 77.0 percent had a university degree, and 18.5 percent had a graduate education. The average job experience of the software developers was 4.01 years. Of the supervisors, 88.2 percent were male, with the average age ranging from 35 to 50. All of the supervisors had at least a university degree and 8.6 percent of them had a graduate education. The supervisors had an average job experience of 20.7 years. The average firm size was 356.17, as measured by the number of employees in the firm. Finally, the average tenure of the relationship between the software developers and supervisors was 1.54 years.

## Measures

*Job performance*. We measured the job performance of the software developers by using a popular seven-item scale that is based on earlier research [35, 59]. As can be seen from the Appendix, we asked the immediate supervisors of the software developers to rate the job performance of the software developers on a seven-point scale ranging from (1) "strongly disagree" to (7) "strongly agree." An example item is "[Software developer] adequately completes assigned duties." The reliability of this measure was 0.94.

*Task autonomy.* We assessed task autonomy by using a popular six-item scale used in earlier studies [36, 37]. The items were specified for the software development tasks. As the Appendix shows, they assessed the extent to which software developers had control over various aspects of their software development activities on a five-point Likert-type scale ranging from (1) "not at all" to (5) "a great deal." The measure included items such as "Do you plan your own software development activities?" and "Can you decide how to go about developing new software?" The reliability score of this measure was 0.92.

*LMX*. The relationship between software developers and their immediate supervisors was assessed based on the seven-item LMX measure, which was suggested as

the most suitable and recommended measure of LMX [19]. As listed in the Appendix, the items were measured on a five-point Likert-type scale with different wordings for the scale points. Sample items include "How well does your leader recognize your potential?" (1 = not at all; 5 = fully) and "How would you characterize your working relationship with your leader?" (1 = extremely ineffective; 5 = extremely effective). This measure had a reliability score of 0.88.

*Informal knowledge utilization processes*: We measured software development firms' informal knowledge utilization processes based on a well-known nine-item scale from Moorman [34]. The items were specified for the software development context. As the Appendix shows, the software developers were asked to indicate whether or not they agreed with the nine statements on a seven-point Likert-type scale ranging from (1) "strongly disagree" to (7) "strongly agree." A sample item is "My division encourages knowledge sharing and utilization in software development." This measure had a reliability of 0.73.

*Formal knowledge utilization processes*. We measured software development firms' formal knowledge utilization processes based on a popular thirteen-item scale [34]. The items were specified for the software development context. As the Appendix shows, the software developers were asked to indicate whether or not they agreed with the thirteen statements on a seven-point Likert-type scale ranging from (1) "strongly disagree" to (7) "strongly agree." A sample item is "My division has formal processes that use knowledge to solve specific problems encountered in software development." This measure had a reliability of 0.81.

*Receiving knowledge*. We measured the knowledge that software developers receive from other software developers in their firms by adapting a popular ten-item scale from the literature [34]. The items were modified for the software development context. As listed in the Appendix, the software developers were asked to indicate whether or not they agreed with the ten statements on a seven-point Likert-type scale ranging from (1) "strongly disagree" to (7) "strongly agree." A sample item is "The knowledge that I receive from other software developers in my company can enrich my basic understanding of the software development process." This measure had a reliability of 0.76.

*Control variables*. We included software developers' gender (1 = male, 2 = female), age (1 = below 25; 2 = 26–34; 3 = 35–50; 4 = above 50), education (1 = primary education, 2 = secondary education, 3 = undergraduate education, 4 = graduate education), job experience (number of years spent in software development), and firm size (number of employees in the firm) as control variables in our analyses and tested all of our hypotheses after controlling for their effects.

## Scale Validities and Reliabilities

Table 1 presents the means, standard deviations, and correlations among the variables. A confirmatory factor analysis including our main variables (i.e., receiving knowledge and job performance) and the moderators (i.e., task autonomy, LMX,

Table 1. Means, Standard Deviations, and Correlations Among Variables

| Variable | Mean | Std. dev. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. Received knowledge | 3.70 | 0.94 | — | | | | | | | | | | |
| 2. Job performance | 4.72 | 1.97 | 0.23** | — | | | | | | | | | |
| 3. Task autonomy | 3.36 | 1.30 | –0.06 | 0.23** | — | | | | | | | | |
| 4. Leader-member exchange (LMX) | 2.73 | 0.99 | 0.03 | 0.20** | 0.18** | — | | | | | | | |
| 5. Informal know. utilization processes | 3.80 | 0.92 | 0.17** | 0.18** | 0.20** | 0.17** | — | | | | | | |
| 6. Formal know. utilization processes | 3.47 | 0.87 | 0.28** | 0.32** | 0.03 | 0.12* | 0.38** | — | | | | | |
| 7. Gender[a] | 1.31 | 0.47 | –0.01 | 0.17** | 0.05 | 0.26** | –0.01 | 0.17** | — | | | | |
| 8. Age[b] | 2.62 | 0.88 | –0.19** | –0.18** | –0.03 | –0.13* | –0.02 | –0.06 | 0.04 | — | | | |
| 9. Education[c] | 3.13 | 0.45 | –0.03** | 0.10 | –0.06 | 0.06 | 0.03 | 0.03 | 0.02 | –0.02 | — | | |
| 10. Job experience[d] | 3.94 | 2.17 | –0.29** | –0.25** | –0.08 | –0.15* | –0.13* | –0.17** | 0.04 | –0.01 | –0.04 | — | |
| 11. Company size[e] | 366.58 | 423.85 | –0.09 | 0.04 | 0.14* | 0.15* | –0.08 | –0.08 | 0.06 | 0.06 | 0.19** | 0.03 | — |

*Notes:* $n = 261$. *$p < 0.05$; **$p < 0.01$.

[a]Gender: 1 = male; 2 = female.

[b]Age: 1 = below 25; 2 = 26-34; 3 = 35-50; 4 = above 50.

[c]Education: 1 = primary education; 2 = secondary education; 3 = undergraduate education; 4 = graduate education (Master's or Ph.D).

[d]Job experience: number of years spent in software development.

[e]Company size: number of employees in the company.

informal knowledge utilization processes, and formal knowledge utilization processes) confirmed the convergent and discriminant validity as well as the unidimensionality of the measures.

The results suggest that the model fits the data well [$\chi^2$ (1259, $n$ = 261) = 2,246.50, $p < 0.001$; $\chi^2/df$ = 1.78; CFI = 0.85; IFI = 0.85; TLI = 0.84; RMSEA = 0.06]. These values are all within the acceptable range for these goodness-of-fit indices [5]. The results also indicated that all path coefficients of the constructs were significant with the lowest $t$-value being 2.11 ($p < 0.05$), thus confirming the convergent validity of the constructs [5]. Finally, the discriminant validity was assessed with the $\chi^2$ difference test, comparing the baseline model with a more restricted model in which the correlation between the constructs is constrained to be 1. A significantly higher $\chi^2$ for the model in which the correlation is restricted would indicate a nonperfect correlation between the constructs and thus a high discriminant validity [5]. A $\chi^2$ test indicated that the difference was significant [$\chi^2$ (15, $n$ = 261) = 3,522.87, $p < 0.001$], providing evidence for discriminant validity.

Although we measured our variables based on the scales developed and used in past research and although our empirical results confirm their reliability and validity, the way variables are measured can threaten the validity of research conclusions [39]. For example, a large number of factors pertaining to the respondents, research settings, research contexts, specific questionnaire items, and many more can lead to measurement error, undermining the effectiveness of the measures [39].

To minimize measurement error and to enhance the effectiveness of our measures, we followed [39] and assured the respondents about the confidentiality of the data. Furthermore, as we list in the Appendix, because our questionnaire items were not socially sensitive in nature, we think the respondents were unlikely to provide socially acceptable answers that would lead to any major measurement bias. In addition, while software developers answered the questions about the independent variable, their supervisors answered the questions about the criterion variable. Finally, we tested our hypotheses using moderated regression models and it is known that measurement errors including common method biases are unlikely to influence the results in moderated regression models such as ours [43].

Podsakoff et al. [39] also suggest that an ineffective measurement might lead the respondents to answer the questions in a particular way. For example, if the questions for the informal knowledge utilization processes are not effective, the variance associated with those questions should be much smaller compared to that associated with the questions pertaining to formal knowledge utilization processes. We compared the variances across the scores for informal and formal knowledge utilization processes and found that while the variance for informal knowledge utilization processes was 0.92, it was 0.87 for formal knowledge utilization processes. These variances did not significantly differ [$F$(261, 2) = 1.59; $p > 0.10$], indicating the effectiveness and robustness of our measures.

## Hypothesis Testing

Our primary focus in this study is the relationship between receiving knowledge from other software developers in the firm and the performance of the knowledge-receiving software developers. At the same time, it is likely that performance is codetermined with receiving knowledge or perhaps performance influences receiving knowledge. Similarly, high performers may be more likely to have better relationships with their superiors (LMX). When there are such endogeneity concerns, there is likely to be a high correlation between the problematic variables (receiving knowledge and LMX) and the error term in the ordinary least squares (OLS) model, which is, in turn, likely to introduce bias into the OLS estimators [28]. In order to address this likely bias and to calculate unbiased estimators, it has been suggested to replace the problematic variables with some instrumental variables that are highly correlated with the problematic variables but uncorrelated with the error terms [28].

Also called instrumental variable methodology, this approach requires the instrumental variables to be both relevant and exogenous [28]. As for "receiving knowledge," we used age and experience as the instrumental variables because there are both theoretical and empirical reasons that they are relevant and exogenous. In fact, it has long been suggested that experience plays an important role in knowledge transfer in organizations [47]. On the other hand, because it has long been known that performance has more to do with the abilities and capabilities of an individual rather than his/her demographic characteristics [56], it is likely that age and experience will be exogenous to predict performance.

We empirically tested the relevance and exogeneity of age and experience as the instrumental variables for receiving knowledge. The results confirm the relevance of our instrumental variables, as indicated by the significant $F$-statistic [6.96 ($p < 0.01$)] in Model I in Table 2. The nonsignificant Hansen $J$-test statistic [0.04 ($p > 0.10$)] in Model I in Table 2 confirms the validity of our instrumental variables for receiving knowledge. Similarly, the nonsignificant Sargan statistic [0.04 ($p > 0.10$)] in Model I in Table 2 confirms the exogeneity of the instrumental variables.

With respect to LMX, we used three instrumental variables including age, experience, and gender because of their relevance and exogeneity based on both theoretical and empirical reasons. In fact, based on dependency and socialization theory, it has long been suggested that compared to mature and experienced employees younger and inexperienced employees are more likely to invest in social exchanges [2]. In addition, it has been suggested that women are more sociable and empathic and therefore have better communication skills than men [13]. As a result, it is likely that compared to male employees, female employees will likely engage more in LMX relationships. On the other hand, because it has long been known that job performance is more related to the abilities and capabilities of an individual rather than his/her demographic characteristics [56], it is likely that these instrumental variables will be exogenous to predict performance.

We empirically tested the relevance and exogeneity of these instrumental variables for LMX. The results confirm the relevance of our instrumental variables, as

Table 2. Regression Results

| Variable | Performance | | | | | |
|---|---|---|---|---|---|---|
| | Model I | | | Model II | | |
| | b | s.e. | *t* | b | s.e. | *t* |
| First-stage models and robustness checks | | | | | | |
| | Receiving knowledge | | | LMX | | |
| Age | –0.21 | 0.06 | –3.37** | –0.16 | 0.07 | –2.48** |
| Experience | –0.12 | 0.03 | –4.91** | –0.08 | 0.03 | –2.79** |
| Gender | | | | 0.58 | 0.13 | 4.62** |
| Instrument relevance: *F*-statistic (*p*-value) | $F(3, 257) = 6.96$ ($p < 0.01$) | | | $F(3, 257) = 6.41$ ($p < 0.01$) | | |
| Instrument validity: Hansen *J*-test statistic (*p*-value) | $\chi^2$ (1) = 0.04 (*ns*) | | | $\chi^2$ (2) = 2.31 (*ns*) | | |
| Instrument exogeneity: Sargan statistic (*p*-value) | $\chi^2$ (1) = 0.04 (*ns*) | | | $\chi^2$ (2) = 2.77 (*ns*) | | |

| Variable | Model III | | | Model IV | | | Model V | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | b | s.e. | *t* | b | s.e. | *t* | b | s.e. | *t* | VIF |
| Second-stage models and hypotheses testing | | | | | | | | | | |
| Education | 0.41 | 0.28 | 1.46 | 0.36 | 0.25 | 1.45 | 0.25 | 0.24 | 1.04 | 1.07 |
| Company size | 0.001 | 0.01 | 0.36 | 0.001 | 0.01 | 0.46 | 0.001 | 0.01 | 0.77 | 1.09 |
| ^Received knowledge | | | | 0.91 | 0.42 | 2.19* | 0.94 | 0.40 | 2.37* | 1.57 |
| Task autonomy (TA) | | | | 0.29 | 0.09 | 3.38** | 0.34 | 0.08 | 3.97** | 1.13 |
| ^LMX | | | | 0.97 | 0.41 | 2.37* | 1.05 | 0.40 | 2.65** | 1.64 |
| Informal knowledge utilization processes (IKUP) | | | | 0.04 | 0.13 | 0.33 | 0.05 | 0.13 | 0.36 | 1.26 |
| Formal knowledge utilization processes (FKUP) | | | | 0.53 | 0.14 | 3.84** | 0.62 | 0.13 | 4.65** | 1.28 |
| ^Received knowledge * TA | | | | | | | 0.57 | 0.25 | 2.27* | 1.10 |
| ^Received knowledge * ^LMX | | | | | | | 1.95 | 0.88 | 2.21* | 1.07 |

| | | | | | | |
|---|---|---|---|---|---|---|
| ^Received knowledge * IKUP | | | -0.41 | 0.48 | -0.85 | 1.55 |
| ^Received knowledge * FKUP | | | 1.65 | 0.47 | 3.49** | 1.54 |
| $R^2$ | 0.01 | 0.24 | 0.31 | | | |
| Adjusted $R^2$ | 0.002 | 0.21 | 0.28 | | | |
| *F*-value | 1.28 | 11.08** | 10.28** | | | |
| Incremental $R^2$ | | 0.23 | 0.08 | | | |
| *F*-value | | 14.87** | 7.03** | | | |

*Notes:* $n = 258$. ^Received knowledge = predicted values from regressing Received knowledge on instrumental variables in Stage 1; ^LMX = predicted values from regressing Leader-member exchange on instrumental variables in Stage 1. * $p < .05$; ** $p < .01$.

indicated by the significant $F$-statistic [6.41 ($p < 0.01$)] in Model II in Table 2. The nonsignificant Hansen $J$-test statistic [2.31 ($p > 0.10$)] in Model II in Table 2 confirms the validity of our instrumental variables for LMX. Similarly, the nonsignificant Sargan statistic [2.77 ($p > 0.10$)] in Model II in Table 2 confirms the exogeneity of the instrumental variables.

After establishing the relevance and exogeneity of the instrumental variables, the first-stage model for "receiving knowledge" included age and experience as the instrumental variables to predict receiving knowledge. Similarly, the first-stage model for LMX included LMX as the dependent variable and age, experience, and gender as the instruments. We calculated the predicted values for receiving knowledge and LMX for each observation based on the first-stage models. As a result, we created new variables for receiving knowledge and LMX based on the predicted values of the first-stage models. Following Kennedy [28], our second-stage model used performance as the dependent variable, and task autonomy, the use of informal knowledge utilization processes in software development, the use of formal knowledge utilization processes in software development, the newly created variables (i.e., predicted values for receiving knowledge and LMX from the first-stage models), and their interactions as predictors. In addition, education and firm size were used as control variables.

We tested the hypotheses by using hierarchical regression analysis [10]. We ran the hierarchical regression models with all main effects first and added the interaction terms later. In order to reduce any potential multicollinearity, we mean-centered all variables [10]. To examine the potential threat of multicollinearity, we also calculated the variance inflation factor (VIF) for each of the regression coefficients. As reported in Table 2, the VIFs were below 10, indicating that multicollinearity was not a likely threat to the parameter estimation [10].

H1 posited that receiving knowledge from other software developers in the firm will have a positive impact on the job performance of the knowledge-receiving software developers. The results in Model V in Table 2 reveal that receiving knowledge had a positive impact on performance [B = 0.94, $t(261) = 2.37$, $p < 0.05$]. As a result, H1 is supported.

H2 suggested that receiving knowledge from other software developers in the firm will have a stronger positive impact on the job performance of the knowledge-receiving software developers who have high rather than low task autonomy. The results in Model V in Table 2 show that this interaction is positive and significant [B = 0.57, $t(261) = 2.27$, $p < 0.05$]. Therefore, H2 is confirmed.

H3 indicated that receiving knowledge from other software developers in the firm will have a stronger positive impact on the job performance of the knowledge-receiving software developers who have high- rather than low-quality LMX relationships with their supervisors. The results in Model V in Table 2 reveal that this interaction is positive and significant [B = 1.95, $t(261) = 2.21$, $p < 0.05$]. As a result, H3 is supported.

H4 posited that receiving knowledge from other software developers in the firm will have a stronger positive impact on the job performance of the knowledge-

receiving software developers when software firms use informal knowledge utilization processes in software development as opposed to when they do not. The results in Model V in Table 2 reveal that this interaction is not significant [B = –0.41, $t(261) = –0.85$, $p > 0.10$], thus rejecting H4.

Finally, H5 suggested that receiving knowledge from other software developers in the firm will have a stronger positive impact on the job performance of the knowledge-receiving software developers when software development firms use formal knowledge utilization processes in software development as opposed to when they do not. The results in Model V in Table 2 reveal that this interaction is positive and significant [B = 1.65, $t(261) = 3.49$, $p < 0.01$]. This result confirms H5.

## Robustness Checks for Omitted Variable Biases

The omitted variables bias is a major issue for most research studies such as ours. In our study, task autonomy and leader–member exchange (LMX) are likely to be proxies for some (unmeasured) organizational factors such as hiring practices, organizational culture, or even the software development methodologies used. Some firms may use agile methodologies, which require significant interaction, and individuals from those firms may report favorable leader–member exchange due to the use of such a methodology.

We applied a series of robustness measures to obtain our estimates. First, we used STATA to apply RESET (regression specification error test) to test whether unknown variables have been omitted from our regression specification [28, p. 76]. The STATA results supported the null hypothesis that the omitted variables bias was in general not a major concern in our parameter estimations [$F(3, 251) = 1.89$, *ns*.].

We then used a more stringent instrumental variable method and applied the Hausman test [28, p. 154], which involves a series of two-stage least squares models. In the first stage, the included explanatory variables are regressed on instrumental variables for the likely omitted variables. The residuals of the first-stage model are then added as extra explanatory variables into the second-stage model regressing the dependent variable (i.e., software development performance in our case) on the included explanatory variables (i.e., received knowledge, task autonomy, LMX, informal knowledge utilization processes, and formal knowledge utilization processes) and the residuals from the first-stage model.

Although the organizational factors that we list above are not measured, we have theoretically relevant instrumental variables for these factors. For example, past research has shown that firm size is an important determinant of hiring practices and that smaller firms tend to use more informal hiring practices than larger firms [3]. Firm size is also related to organizational culture. In particular, small firms are found to be significantly more supportive, participative, and performance-oriented than medium-size or large organizations [11].

Moreover, the use of agile software development methods involves significant interaction between the software developers and their supervisors [18, 41]. However,

as can be seen from our theoretical model, we already have a theoretical variable for the use of informal knowledge utilization processes in software development. Instead of using this variable as a theoretical variable, we treated it as an instrumental variable for firms' use of agile software development practices and applied the Hausman test [28, p. 154] to test the effects of omitting the unmeasured organizational factors that we list above on our parameter estimates.

Using firm size and the use of informal knowledge utilization processes as instrumental variables and applying the two-stage methodology [28, p. 154], we found that our explanatory variables were not correlated with the error terms as indicated by the nonsignificant Hausman test [$F(2, 252) = 1.47$, *ns*.]. The results of RESET and Hausman tests consistently suggest that although organizational factors and software development methods are related to software development performance, omitting them does not impact the parameter estimations in our case because our explanatory variables have their own unique impacts on this performance independent of the organizational factors and software development practices.

## Discussion and Conclusion

We drew on task closure theory [46] to study how the knowledge that software developers receive from other software developers in their firm might impact their performance. We further integrated this theory with three complementary theories about knowledge application contexts to suggest that while software developers need to focus on their immediate software development tasks, they need to constantly interact with their supervisors and perform their tasks within a larger institutional setting with both implicit and explicit rules as well as standards, documents, and metrics.

Our empirical study showed that receiving knowledge from other software developers in the firm can indeed increase the performance of the knowledge-receiving software developers. It also suggested that this relationship was stronger when the software developers had the freedom to incorporate into their software development tasks the knowledge that they received from their peers. In addition, the results showed that the relationship was stronger when software developers had high- rather than low-quality LMX relationships with their supervisors. Moreover, the results suggested that the relationship was stronger when software development firms adopted more formal but not informal knowledge utilization processes.

Expanding upon past research that emphasized the role of institutional interventions in knowledge sharing and utilization, we distinguished between informal and formal organizational interventions and found that formal knowledge utilization processes ($p < 0.01$) had a stronger impact than informal knowledge utilization processes ($p > 0.10$) on software developers' performance. In addition, formal ($p < 0.01$) but not informal ($p > 0.10$) knowledge utilization processes moderated the relationship between received knowledge and performance in software development. Taken together, these results offer an empirical test for the informal and formal information integration mechanisms suggested by Grant [20] and provide an

integrated treatment of those mechanisms. They indicate that merely urging software developers to share and utilize knowledge in a firm may not be enough to facilitate knowledge sharing and utilization. Instead, more formal knowledge utilization interventions might be necessary to facilitate knowledge sharing and application. Because software development is considered to be complex and also to require stringent performance [12, 54], formal knowledge utilization processes appear to be more applicable than informal knowledge utilization processes in the context of software development because formal organizational routines can be more relevant in complex circumstances with stringent performance requirements [20]. Future research should compare these processes in other contexts with varying levels of complexities and performance requirements.

## Research Implications

This study makes several research contributions. First, earlier studies on software process improvement have recognized that while technology- and/or process-oriented approaches to improve the software development process are important, an in-depth understanding of the human side of the software development process will also be very useful [33]. Given the significance of knowledge application in the software development process and the importance of studying software developers as knowledge recipients in this process [12, 51], we studied how software developers benefit from the knowledge that they receive from other software developers in their firm. This contributes to research on software process improvement in terms of offering a better understanding of knowledge application in software development.

In addition, this study suggested that software developers' utilization of the knowledge that they receive from their peers differed with respect to the task, social, and institutional dimensions of the software development context. These findings further refine and clarify our current theoretical understanding of the performance implications of knowledge sharing and their boundary conditions within the context of software development and contribute to research with respect to reducing the gap between accumulated knowledge and the actual utilization of that knowledge [1, 20].

Moreover, we integrated task closure theory with three complementary theories related to knowledge application contexts to study how software developers apply the knowledge that they receive from other software developers in their firms and thus bring closure in their knowledge sharing activities. This integrated and fine-grained framework contributes to research by allowing for a more specific articulation of the processes through which software developers can utilize the knowledge that they receive from other software developers in the firm.

## Practical Implications

This study also makes several practical contributions. For example, it showed how software developers benefit from the knowledge that they receive from other software

developers in their firm. This helps improve the software development process. In addition, our results about the moderating roles of the task, social, and institutional dimensions of the software development context can help firms increase knowledge utilization in software development. Moreover, the results about the moderating role of LMX can also provide a foundation for designing and developing LMX training models so that supervisors can develop and maintain high-quality LMX relationships with their subordinates [19]. Finally, it has been widely recognized that software development is a performance-driven and highly stressful activity [8]. Thus, our study also helps enhance the job performance of software developers.

## Limitations

This study has several limitations. First, we focused primarily on how the knowledge that software developers receive from other software developers in the company might impact their performance, insofar as it is widely recognized that individual performance plays a major role in knowledge-driven value creation activities such as software development [8, 38]. Future research can look at how the knowledge that teams, business units or organizations receive from their respective counterparts impacts their respective performance.

Second, we focused only on the task, social, and institutional dimensions of the software development context and studied their moderating roles in the relationship between receiving knowledge and performance. Because a single study will not be able to cover all the possible factors moderating this complex relationship, future studies can identify other personal and nonpersonal moderators and investigate their moderating roles in this relationship.

Third, similar to other survey-based studies using perceptual scales, we studied the relationship between knowledge-receiving and performance in software development based on a survey using perceptual measures. As we stated in the measures section above, we used existing scales and applied both procedural and statistical measures to ensure their effectiveness. Future studies can conduct more in-depth cognitive studies to identify some of the reasons that software developers use or do not use the knowledge that they receive from their peers in their company.

Fourth, software development performance is a multidimensional activity that can be affected not only by individual- but also organizational-level factors such as firms' hiring practices, organizational culture, and software development practices. Our additional robustness checks showed that omitting such organizational factors was not a significant concern within the context of our study. However, it might be an important concern in other contexts and with the use of other sets of explanatory variables. Hence, our results should be interpreted in light of this potential bias and future studies should control for this potential bias in other study contexts.

Finally, similar to past research [24], our control variables were mostly demographic variables such as age, gender, education, job experience, and firm size. However, our dependent variable, performance in software development, is likely to

be influenced by numerous important nondemographic factors. For example, individuals in analyst or project/team lead positions may receive and utilize knowledge differently from individuals in pure programming roles. As we indicated in the methodology section of our paper, in order to control for common method bias, we surveyed software developers and their immediate supervisors, while software developers answered questions about the independent variables, their immediate supervisors answered questions about the dependent variable. As a result, our main sample included individual software developers without any project/team role while the team leaders assessed the performance of their subordinates in the team. Although our research design allows us to indirectly control for the effect of the job role of software developers, future research should replicate our study by directly controlling for this variable. In addition, organizational factors such as organizational software capability maturity would have a significant influence on the nature of knowledge transfer practices used and their effectiveness. Moreover, several software development factors such as method used, task complexity, development or maintenance work, and so on, as well as individual factors such as firm and job tenures are likely to influence software developers' knowledge sharing and performance. Thus, our findings should be interpreted in light of this limitation and future research should control for these important control variables.

## Conclusions

Software development is a knowledge-intensive activity that requires extensive knowledge sharing among software developers. We used task closure theory [46] to study the impact of the knowledge that software developers receive from other software developers in the firm on their performance. Integrating this theory with three contextual theories, we also studied how the task, social, and institutional contexts in which software developers work moderate this relationship. The results in general show that software developers benefit from receiving knowledge from other software developers in the firm. Moreover, this relationship was stronger when the software developers had high task autonomy, maintained high-quality social exchanges with their supervisors, and worked in firms with formal knowledge utilization processes. The results contribute to a better understanding of knowledge utilization in software development. Moreover, they help software development firms to improve the software development process through enhancing knowledge utilization in this process.

## References

1. Alavi, M., and Leidner, D.E. Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues. *MIS Quarterly*, *25*, 1 (2001), 107–136.

2. Ashford, S.J., and Black, J.S. Proactivity during organizational entry: The role of desire for control. *Journal of Applied Psychology*, *81*, 2 (1996), 199–214.

3. Barber, A.E.; Wesson, M.J.; Roberson, Q.M.; and Taylor, M.S. A tale of two job markets: Organizational size and its effects on hiring practices and job search behavior. *Personnel Psychology*, *52*, 4 (1999), 841–868.

4. Briggs, R.O.; Kolfschoten, G.L.; de Vreede, G.; Lukosch, S.; and Albrecht C.C. Facilitator-in-a-box: Process support applications to help practitioners realize the potential of collaboration technology. *Journal of Management Information Systems*, *29*, 4 (2013), 159–194.

5. Byrne, B.M. *Structural Equation Modeling with AMOS: Basic Concepts, Applications, and Programming*. New York: Routledge, 2013.

6. Carmel, E.; Espinosa, J.A.; and Dubinsky, Y. "Follow the sun" workflow in global software development. *Journal of Management Information Systems, 27*, 1 (2010), 17–37.

7. Chai, S.; Das, S.; and Rao, H.R. Factors affecting bloggers' knowledge sharing: An investigation across gender. *Journal of Management Information Systems*, *28*, 3 (2011), 309–342.

8. Chilton, M.; Hardgrave, B.C.; and Armstrong, D.J. Person-job cognitive style fit for software developers: The effect on strain and performance. *Journal of Management Information Systems*, *22*, 2 (2005), 193–226.

9. Choi, S.Y.; Lee, H.; and Yoo, Y. The impact of information technology and transactive memory systems on knowledge sharing, application and team performance: A field study. *MIS Quarterly*, *34*, 4 (2010), 855–870.

10. Cohen, J.; Cohen, P.; West, S.G.; and Aiken, L.S. *Applied Multiple Regression/ Correlations Analysis for the Behavioral Sciences*. 3rd ed. Mahwah, NJ: Erlbaum, 2003.

11. Connell, J. Influence of firm size on organizational culture and employee morale. *Journal of Management Research, 1*, 4 (2001), 220–232.

12. Cross, R., and Sproull, L. More than an answer: Information relationships for actionable knowledge. *Organization Science*, *15*, 4 (2004), 446–462.

13. Deery, S.; Iverson, R.; and Walsh, J. Work relationships in telephone call centers: Understanding emotional exhaustion and employee withdrawal. *Journal of Management* Studies, *39*, 5 (2002), 471–496.

14. Dulebohn, J.H.; Bommer, W.H.; Liden, R.C.; Brouer, R.L.; and Ferris, G.R. A meta-analysis of antecedents and consequences of leader-member exchange: Integrating the past with an eye toward the future. *Journal of Management*, *38*, 6 (2012), 1715–1759.

15. Elie-dit-cosaque, C.; Pallud, J.; and Kalika, M. The influence of individual, contextual, and social factors on perceived behavioral control of information technology: A field theory approach. *Journal of Management Information Systems*, *28*, 3 (2012), 201–234.

16. Fruhling, A., and De Vreede, G. Field experiences with eXtreme programming: Developing an emergency response system. *Journal of Management Information Systems*, *22*, 4 (2006), 39–68.

17. Gerstner, C.R., and Day, D.V. Meta-analytic review of leader-member exchange theory: Correlates and construct issues. *Journal of Applied Psychology*, *82*, 6 (1997), 827–844.

18. Gopal, A., and Gosain, S. The role of organizational controls and boundary spanning in software development outsourcing: Implications for project performance. *Information Systems Research*, *21*, 4 (2010), 960–982.

19. Graen, G.B., and Uhl-Bien, M. Relationship-based approach to leadership: Development of leader-member exchange (LMX) theory of leadership over 25 years: Applying a multi-level multi-domain perspective. *Leadership Quarterly*, *6*, 2 (1995), 219–247.

20. Grant, R.M. Prospering in dynamically-competitive environments: Organizational capability as knowledge integration. *Organization Science*, *7*, 4 (1996), 375–387.

21. Hackman, J.R., and Oldham, G.R. Work redesign and motivation. *Professional Psychology*, *11*, 3 (1980), 445–455.

22. Hardgrave, B.C.; Davis, F.D.; and Riemenschneider, C.K. Investigating determinants of software developers' intentions to follow methodologies. *Journal of Management Information Systems*, *20*, 1 (2003), 123–151.

23. He, J.; Butler, B.S.; and King, W.R. Team cognition: Development and evolution in software project teams. *Journal of Management Information Systems*, *24*, 2 (2007), 261–292.

24. Jarvenpaa, S.L., and Staples, D.S. Exploring perceptions of organizational ownership of information and expertise. *Journal of Management Information Systems*, *18*, 1 (2001), 151–183.

25. Kane, A.A. Unlocking knowledge transfer potential: Knowledge demonstrability and superordinate social identity. *Organization Science*, *21*, 3 (2010), 643–660.

26. Ke, W.L.; Tan, C.H.; Sia C.L.; and Wei K.K. Inducing intrinsic motivation to explore the enterprise system: The supremacy of organizational levers. *Journal of Management Information Systems*, *29*, 3 (2012), 257–290.

27. Keith, M.; Demirkan, H.; and Goul, M. Service-oriented methodology for systems development. *Journal of Management Information Systems*, *30*, 1 (2013), 227–260.

28. Kennedy, P. *A Guide to Econometrics*. 6th ed. Malden, MA: Wiley-Blackwell, 2008.

29. Langfred, C.W., and Moye, N.A. Effects of task autonomy on performance: An extended model considering motivational, informational, and structural mechanisms. *Journal of Applied Psychology*, *89*, 6 (2004), 934–945.

30. Lee, G., and Xia, W. Towards agile: An integrated analysis of quantitative and qualitative field data on software development agility. *MIS Quarterly*, *34*, 1 (2010), 87–114.

31. Lee, J.S.; Keil, M.; and Kasi, V. The effect of an initial budget and schedule goal on software project escalation. *Journal of Management Information Systems*, *29*, 1 (2012), 53–78.

32. Mastrogiacomo, S.; Missonier, S.; and Bonazzi, R. Talk before it's too late: Reconsidering the role of conversation in information systems project management. *Journal of Management Information Systems*, *31*, 1 (2014), 44–78.

33. Mehra A., and Mookerjee V. Human capital development for programmers using open source software. *MIS Quarterly*, *36*, 1 (2012), 107–122.

34. Moorman, C. Organizational market information processes: Cultural antecedents and new product outcomes. *Journal of Marketing Research*, *32*, 3 (1995), 318–335.

35. Ozer, M. Personal and task-related moderators of leader-member exchange among software developers. *Journal of Applied Psychology*, *93*, 5 (2008), 1174–1182.

36. Ozer, M. A moderated mediation model of the relationship between organizational citizenship behaviors and job performance. *Journal of Applied Psychology*, *96*, 6 (2011), 1328–1336.

37. Parker, S.K., and Carolyn M. Axtell, C.M. Seeing another viewpoint: Antecedents and outcomes of employee perspective taking. *Academy of Management Journal*, *44*, 6 (2001), 1085–1100.

38. Ply, J.K.; Moore, J.E.; Williams, C.K.; and Thatcher, J.B. IS employee attitude and perceptions at varying levels of software process maturity. *MIS Quarterly*, *36*, 2 (2012), 601–624.

39. Podsakoff, P.M.; MacKenzie, S.B.; Lee, J.Y.; and Podsakoff, N.P. Common method biases in behavioral research: A critical review of the literature and recommended remedies. *Journal of Applied Psychology*, *88*, 5 (2003), 879–903.

40. Quigley, N.; Tesluk, P.E.; Locke, E.A.; and Bartol, K.M. The effects of incentives and individual differences on knowledge sharing and performance effectiveness. *Organization Science*, *18*, 1 (2007), 71–88.

41. Ramesh, B.; Mohan, K.; and Cao, L. Ambidexterity in agile distributed software development: An empirical investigation. *Information Systems Research*, *23*, 2 (2012), 323–339.

42. Schmidt, R.; Lyytinen, K.; Keil, M.; and Cule, P. Identifying software project risks: An international study. *Journal of Management Information Systems*, *17*, 4 (2001), 5–36.

43. Siemsen, E.; Roth, A.; and Oliveira P. Common method bias in regression models with linear, quadratic, and interaction effects. *Organizational Research Methods*, *13*, 3 (2010), 456–476.

44. Singh, P.V., and Tan, Y. Developer heterogeneity and formation of communication networks in open source software projects. *Journal of Management Information Systems*, *27*, 3 (2010), 179–210.

45. Srivastava, S.C., and Teo, T.S.H. Contract performance in offshore systems development: Role of control mechanisms. *Journal of Management Information Systems*, *29*, 1 (2012), 115–158.

46. Straub, D.W., and Karahanna, E. Knowledge worker communications and recipient availability: Toward a task closure explanation of media choice. *Organization Science*, *9*, 2 (1998), 1–16.

47. Swap, W.C.; Leonard, D.A.; Shields, M.; and Abrams, L. Using mentoring and storytelling to transfer knowledge in the workplace. *Journal of Management Information Systems*, *18*, 1 (2001), 95–114.

48. Te'eni, D. A cognitive-affective model of organizational communication for designing IT. *MIS Quarterly*, *25*, 2 (2001), 251–312.

49. Tiwana, A. Systems development ambidexterity: Explaining the complementary and substitutive roles of formal and informal controls. *Journal of Management Information Systems*, *27*, 2 (2010), 87–126.

50. Tiwana, A. Novelty-knowledge alignment: A theory of design convergence in systems development. *Journal of Management Information Systems*, *29*, 1 (2012), 15–52.

51. Tiwana, A., and McLean, E.R. Expertise integration and creativity in information systems development. *Journal of Management Information Systems*, *22*, 1 (2005), 13–44.

52. US. Department of Commerce Bureau of Economic Analysis. *Gross Domestic Product (GDP)*. http://www.bea.gov/newsreleases/national/gdp/2015/pdf/gdp1q15_adv.pdf (accessed on April 30, 2015).

53. Vidgen, R., and Wang, X. Coevolving systems and the organization of agile software development. *Information Systems Research*, *20*, 3 (2009), 355–376.

54. Vlas, R.E., and Robinson, W.N. Two rule-based natural language strategies for requirements discovery and classification in open source software development projects. *Journal of Management Information Systems*, *28*, 4 (2012), 11–38.

55. von Krogh, G.; Haefliger, S.; Spaeth, S.; and Wallin, M. Carrots and rainbows: Motivation and social practice in open source software development. *MIS Quarterly*, *36*, 2 (2012), 649–676.

56. Wade, M.R., and Parent, M. Relationships between job skills and performance: A study of webmasters. *Journal of Management Information Systems*, *18*, 3 (2001), 71–96.

57. Weiyin, H.; Thong, J.L.; Chasalow, L.C.; and Dhillon, G. User acceptance of agile information systems: A model and empirical test. *Journal of Management Information Systems*, *28*, 1 (2011), 235–272.

58. Welbourne, T.M.; Johnson, D.E.; and Erez, A. The role-based performance scale: Validity analysis of a theory-based measure. *Academy of Management Journal*, *41*, 5 (1998), 540–555.

59. Williams, L.J., and Anderson, S.E. Job satisfaction and organizational commitment as predictors of organizational citizenship and in-role behaviors. *Journal of Management*, *17*, 3 (1991), 601–617.

60. Xu, P., and Ramesh, B. Software process tailoring: An empirical investigation. *Journal of Management Information Systems*, *24*, 2 (2007), 293–328.

## Appendix: Questionnaire Items

*Performance*: Immediate supervisors of the software developers were asked to rate the job performance of the software developers on a seven-point scale ranging from (1) "strongly disagree" to (7) "strongly agree." Adapted from [35, 59].

1. [Software developer] adequately completes assigned duties.
2. [Software developer] fulfills responsibilities specified in job description.
3. [Software developer] performs tasks that are expected of him/her.
4. [Software developer] meets formal performance requirements of the job.
5. [Software developer] engages in activities that will directly affect his/her performance evaluation.
6. [Software developer] neglects aspects of the job he/she is obligated to perform. [REVERSE CODED].
7. [Software developer] fails to perform essential duties. [REVERSE CODED].

*Task autonomy*: The software developers were asked to indicate the extent to which they had control over various aspects of their software development activities on a five-point scale ranging from (1) "not at all" to (5) "a great deal." Adapted from [36, 37].

1. Can you control how much you work?
2. Can you vary how you develop new software?
3. Do you plan your own software development activities?
4. Can you control the quality of your software development activities?
5. Can you decide how to go about developing new software?
6. Can you choose the methods to use in designing new software?

*LMX*: The software developers were asked to indicate their relationships with their supervisors on a five-point scale with different endpoints for each question as listed below. Adapted from [19].

1. Do you know where you stand with your leader . . . do you usually know how satisfied your leader is with what you do?
   (1 = Rarely, 2 = Occasionally, 3 = Sometimes, 4 = Fairly Often, 5 = Very Often).
2. How well does your leader understand your job problems and needs?
   (1 = Not a Bit, 2 = A Little, 3 = A Fair Amount, 4 = Quite a Bit, 5 = A Great Deal).
3. How well does your leader recognize your potential?
   (1 = Not at All, 2 = A Little, 3 = Moderately, 4 = Mostly, 5 = Fully).
4. Regardless of how much formal authority he/she has built into his/her position, what are the chances that your leader would use his/her power to help you solve problems in your work?
   (1 = None, 2 = Small, 3 = Moderate, 4 = High, 5 = Very High).
5. Again, regardless of the amount of formal authority your leader has, what are the chances that he/she would "bail you out," at his/her expense?
   (1 = None, 2 = Small, 3 = Moderate, 4 = High, 5 = Very High).
6. I have enough confidence in my leader that I would defend and justify his/her decision if he/she were not present to do so.
   (1 = Strongly Disagree, 2 = Disagree, 3 = Neutral, 4 = Agree, 5 = Strongly Agree).
7. How would you characterize your working relationship with your leader?
   (1 = Extremely Ineffective, 2 = Worse Than Average, 3 = Average, 4 = Better Than Average, 5 = Extremely Effective).

*Informal knowledge utilization processes*: The software developers were asked to indicate whether or not they agreed with the following nine statements on a seven-point scale ranging from (1) "strongly disagree" to (7) "strongly agree." Adapted from [34].

1. My division encourages summarizing knowledge to reduce its complexity in software development.

2. My division encourages disagreeing and challenging others' opinions in software development.
3. My division encourages knowledge generation in software development.
4. My division encourages organizing knowledge in meaningful ways for software development.
5. My division encourages knowledge sharing and utilization in software development.
6. My division values knowledge as an aid to software development.
7. My division views new knowledge as disruptive to software development. [REVERSE CODED]
8. My division devalues the role of knowledge providers in software development. [REVERSE CODED].
9. My division structures jobs so that knowledge providers play a role in software development.

*Formal knowledge utilization processes*: The software developers were asked to indicate whether or not they agreed with the following thirteen statements on a seven-point scale ranging from (1) "strongly disagree" to (7) "strongly agree." Adapted from [34].

1. My division has formal processes for carefully evaluating various software development alternatives.
2. My division has formal processes that rely heavily upon knowledge to make decisions relating to software development.
3. My division has formal processes that use knowledge to solve specific problems encountered in software development.
4. My division has formal processes that provide knowledge to effectively develop software.
5. My division has formal processes that provide clear directions on software development.
6. My division has formal processes that give knowledge to all functions regarding their role in software development.
7. My division has formal processes that formally evaluate the effectiveness of software development.
8. My division has formal processes that provide informal feedback regarding the effectiveness of software development.
9. My division has formal processes that provide feedback to software developers regarding the outcomes of their software development activities.
10. My division has formal processes that constructively evaluate the outcomes of software development.
11. My division has formal processes that encourage software developers to understand the reasons for their mistakes throughout the software development process.

12. My division has formal processes that integrate knowledge from a variety of sources when developing software.
13. My division has formal processes that ensure that all knowledge sources are considered in software development (not only those that support the preferred action).

*Received knowledge*: The software developers were asked to indicate whether or not they agreed with the following ten statements on a seven-point scale ranging from (1) "strongly disagree" to (7) "strongly agree." Adapted from [34].

1. The knowledge that I receive from other software developers in my company can enrich my basic understanding of the software development process.
2. The way I think about the software development process will be very different if I do not receive any knowledge from other software developers in my company.
3. I constantly think about the software development process.
4. Receiving knowledge from the other software developers in the company really enlightens my understanding of the software development process.
5. The knowledge that I receive from other software developers in my company reduces my uncertainty about the software development process.
6. The knowledge that I receive from other software developers in my company helps me identify aspects of the software development process that can otherwise go unnoticed.
7. My ability to make software development decisions will be diminished without the knowledge that I receive from other software developers in my company.
8. My software development decisions really do not require any knowledge from other software developers in my company. [REVERSE CODED].
9. I use the knowledge that I receive from other software developers in my company to make specific software development decisions.
10. Without the knowledge that I receive from other software developers in my company, my software development decisions will be very different.

*Control variables*: We ask the software developers to indicate their:

Gender (1 = male, 2 = female)
Age (1 = below 25; 2 = 26–34; 3 = 35–50, 4 = above 50)
Education (1= primary education, 2 = secondary education, 3 = undergraduate education, 4 = graduate education)
Job experience (number of years spent in software development)
Firm size (number of employees in the firm)