# Task Environment Complexity, Global Team Dispersion, Process Capabilities, and Coordination in Software Development

Gwanhoo Lee, J. Alberto Espinosa, and William H. DeLone

**Abstract**—Software development teams are increasingly global. Team members are separated by multiple boundaries such as geographic location, time zone, culture, and organization, presenting substantial coordination challenges. Global software development becomes even more challenging when user requirements change dynamically. However, little empirical research has investigated how team dispersion across multiple boundaries and user requirements dynamism, which collectively increase task environment complexity, influence team coordination and software development success in the global context. Further, we have a limited understanding of how software process capabilities such as rigor, standardization, agility, and customizability mitigate the negative effects of global team dispersion and user requirements dynamism. To address these important issues, we test a set of relevant hypotheses using field survey data obtained from both project managers and stakeholders. Our results show that global team dispersion and user requirements dynamism have a negative effect on coordination effectiveness. We find that the negative effect of global team dispersion on coordination effectiveness decreases as process standardization increases and that the negative effect of user requirements dynamism on coordination effectiveness decreases as process agility increases. We find that coordination effectiveness has a positive effect on global software development success in terms of both process and product aspects.

**Index Terms**—Global boundaries, global software development, user requirements dynamism, software process capability, task environment complexity, team coordination, team dispersion

## 1 Introduction

Software development has become much more geographically dispersed in recent years in part due to globalization and outsourcing [1]. A recent Gartner report [2] indicates that more than 90 percent of Fortune 500 companies use external resources for IT services delivery and that 31 percent of IT spending by companies in 2010 was on external services. One software engineer of a large IT services company we interviewed said "It is nearly impossible these days to find a software team that is completely collocated." However, globally dispersed software teams may deal with special challenges as they are often divided by multiple boundaries, such as geographic location, time zones, culture, and organization [3], [4]. In support of this view, an IBM report [5] stated "Going global offers many benefits. However, distributed teams face more challenges than collocated teams." Carrying out software development work across multiple global team boundaries is difficult because of the substantial coordination challenges associated with bridging these boundaries to get the work done. This problem is particularly severe in software development because of the dynamic nature of user requirements, which imposes the need to keep track of and act upon frequent changes affecting the final software.

• *The authors are with the Kogod School of Business, American University, 4400 Massachusetts Avenue, NW, Washington, DC 20016-8044.*

*Manuscript received 7 Dec. 2012; revised 1 May 2013; accepted 13 Aug. 2013; published online 19 Aug. 2013.*
*Recommended for acceptance by L. Williams.*
*For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2012-12-0362.*
*Digital Object Identifier no. 10.1109/TSE.2013.40.*

In this study, we argue that the need for more coordination when user requirements change often, coupled with the coordination barriers imposed by global team dispersion across multiple boundaries, increase the task environment complexity. While task complexity has been studied extensively [6], [7], [8], [9], [10], [11], [12], the complexity of the *task environment* has received little attention in the literature [13], [14], [15]. In particular, we argue that because software development involves highly dependent activities [16], and because the essence of coordination is the management of these dependencies [17], this additional complexity in the task environment makes it more difficult to coordinate software work effectively. While there are many factors that can contribute to increased complexity in the task environment, two are particularly important to global software teams—team dispersion across global boundaries and user requirements dynamism.

Global teams are dispersed across multiple boundaries, which increases the amount of information that team members need to consider when working with each other [18]. Moreover, these team boundaries (e.g., distance, time, cultural, and organizational) often coexist [3], [19], making the effects of individual boundaries difficult to observe. For example, some empirical studies of global teams have investigated the effects of a single boundary such as geographic distance [18], [20], [21], modeling it as a binary variable (e.g., colocated versus distributed) or as the number of locations, without controlling for effects of other global boundaries. Similarly, studies of team coordination across time zones have modeled time separation as a time span difference or number of time zones [16]. In this study,

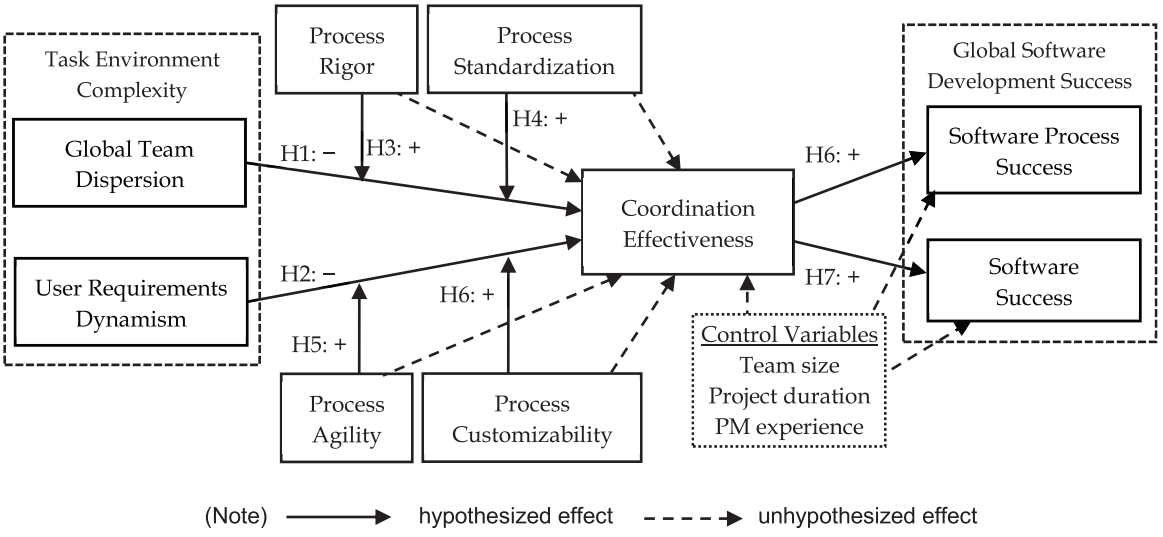



Fig. 1. Research model.

we argue that it is not necessarily the presence of one or more boundaries that makes a task environment complex for coordination and collaboration, but how team members are dispersed across various global boundaries. We argue that coordination challenges are greater when members need to cross more global boundaries to collaborate and when members are more dispersed across these boundaries. Conversely, coordination costs are expected to be lower when more work can be done without crossing such boundaries [3].

The other component of task environment complexity results from user requirements dynamism in software development [15]. User requirements are dynamic and volatile when uncertainties are involved in understanding the functional scope of a software system [22]. Software development generally involves frequent requirements changes. Studies have shown that around 25 percent of the initial requirements change in medium-sized software projects and over 35 percent in large-scale projects [23]. Thus, user requirements changes during the software development life cycle are the norm, rather than the exception and these changes are likely to be more salient in a global development environment. We discuss these two components of task environment complexity further in Section 2.1.

While we argue that global boundary dispersion and user requirements dynamism increase the complexity of the software development environment and, therefore, negatively influence coordination effectiveness, we also argue that such detrimental effects can be mitigated by appropriate software process capabilities. Prior research suggests that effective software processes contribute to software development success [24]. A software process is defined as a set of activities performed during the development of a software system [25] and these activities involve interactions among people, technology, methods, and procedures [26]. Among different types of software process capabilities, we identify *rigor, standardization, agility, and customizability* as key capabilities for global software development from the prior literature and our preliminary field interviews [27]. Standardized and rigorous software processes may offset some of the negative effects of team dispersion on development performance, whereas ad hoc, ill-defined processes may make software development fall into disarray with increased dispersion. On the other hand, agile and customizable software processes may help teams cope with user requirements dynamism as such processes can lower cost and shorten time in responding to user change requests. To the best of our knowledge, these moderating effects have not been investigated.

In sum, our research contributes to the software engineering literature by studying the effect of software development task environment complexity (rather than task complexity) resulting from global team dispersion across multiple boundaries and user requirements dynamism on software development coordination effectiveness. Our research also contributes by investigating the mitigating roles of software process capabilities in coping with increased task environment complexity. Consequently, our study seeks to answer the following important questions, which have not been adequately answered by prior research:

1. *What are the effects of task environment complexity on coordination effectiveness in global software development? More specifically:*
   a. *What are the effects of global team dispersion across multiple boundaries?*
   b. *What are the effects of user requirements dynamism?*
2. *What are the moderating effects of software process capabilities—i.e., rigor, standardization, agility, and customizability—on the effects of task environment complexity on coordination effectiveness of global software development?*

## 2 Theory and Hypotheses

The research model and hypotheses for our study are shown in Fig. 1. The hypotheses and their theoretical background and underpinnings are presented in the following sections.

### 2.1 Task Environment Complexity

We distinguish between task complexity and task environment complexity. The former refers to the inherent complexity of the task itself, regardless of the nature of the collaboration environment in which the task is situated. Task complexity in software development increases as the number of task components (e.g., software modules) [28] and their relatedness increase [29]. In contrast, task environment complexity refers to the complexity of the collaboration environment in which the task is situated, regardless of the nature of the task itself. Task environmental complexity in software development increases as the development team is more geographically dispersed and culturally diverse, and as user requirements change more frequently. While task complexity increases the cognitive demands on the team members who are performing the tasks because they need to process more information cues to get the job done [9], [12], it does not necessarily increase coordination costs with other members. Conversely, higher task environment complexity increases the number of information cues that need to be processed to work with other members on the task.

Coordination theory defines coordination as the "management of dependencies among task activities" [30]. Task activities that are the responsibility of a single individual are affected by the complexity of the task, but are not affected so much by the complexity of the task environment and, therefore, have very little impact on coordination effectiveness. Conversely, interdependent task activities that are assigned to more than one member will require coordination to manage these dependencies. Prior research has studied software task complexity extensively [28], [29], [31], but to the best of our knowledge, very few studies have examined the complexity of the software task environment [14], which is prevalent in global software teamwork. From the complexity theory perspective, task environment complexity comprises *structural complexity* and *dynamic complexity* [15]. In the context of global software development, team dispersion represents an important aspect of structural complexity and user requirements dynamism represents an important aspect of dynamic complexity.

### 2.2 Effect of Global Team Dispersion on Coordination Effectiveness

Team members working in a software development project are often separated by boundaries such as geographic distance, time zone, culture, and organization [3]. We argue that it is not so much the presence of a particular boundary or the number of boundaries present that affect team coordination effectiveness, but how members are distributed across these boundaries. Members of a team without boundaries become easily familiar with their task context and each other's work routines because they interact with each other frequently. But as team members are more evenly distributed across locations, time zones, cultures, and organizations, more boundaries need to be bridged to manage their task dependencies. A team is more evenly distributed across a boundary when its members are more widely dispersed across that boundary. For example, a team with five members in one site and seven members in the other site is more evenly distributed across a geographic boundary than a team with one member in one site and eleven members in the other site, which results in more coordination effort.

Dependencies among task activities can be pooled, sequential, or reciprocal [32] and they can be coordinated either mechanistically (i.e., by program or by plan, e.g., project schedules, plans, specifications, procedures, etc.) or organically (i.e., by interacting and communicating, e.g., meetings, one-on-one discussions, etc.) [33], [34]. Furthermore, a substantial amount of organic coordination in software development takes place informally through spontaneous encounters [35]. The presence of a global boundary of any type between any two team members makes it more difficult to coordinate formally and less likely for members to meet informally to coordinate [36]. So, naturally, more dispersion across global boundaries makes the task environment more complex because it increases the number of boundaries that need to be bridged to collaborate and the number of interdependent information cues that team members need to process to carry out the task [3], [4], [12], [37], [38]. Increased task dependencies across boundaries are associated with increased demands for coordination and communication effort [9]. For example, the coordination challenges in a software development project with four sites, three time zones, and two different cultures, with members evenly distributed across them will be more substantial than when members are concentrated in one site, one time zone, and share the same culture and organization affiliation. But this condition has not been incorporated in the prior studies of global software teams.

This study utilizes and tests a multiboundary global team dispersion measure in the context of global software development. We conceptualize this measure from the perspective of the structural complexity of the task environment, rather than based on single boundary measures. While various global team boundaries have been discussed in the literature, four of them have been consistently suggested as most important in team research—geographic location, time zone, culture, and organization [3], [4], [39], which our field interviews also confirmed. Therefore, this study focuses on these four boundaries.

Much attention has been paid to the effects of individual boundaries in prior research [20], [21], [40], but we argue that it is the simultaneous presence of multiple boundaries that makes global software development more challenging. Further, prior research has employed the number of boundaries (e.g., number of time zones, number of geographic locations, etc.) to measure the extent to which a team is distributed [16], [37]. But we argue that it is the distribution of members across these boundaries that significantly impacts team coordination effectiveness [38]. Each boundary between any two members creates a discontinuity between them, which increases the coordination costs incurred to bridge that boundary [41]. As team members become more evenly dispersed across these boundaries, the number of boundaries between members increases, thus increasing the coordination challenges. When a team is heavily concentrated in one location, time zone, culture, or organization, even if multiple boundaries exist in the team, it is relatively easy for the team to

coordinate tasks because most of the work is carried out in the same place, time zone, or organization, and thus, task dependencies can be relatively easily managed [42]. Conversely, when a team is more evenly distributed across boundaries, more members need to cross boundaries to coordinate their task activities, which imposes additional external constraints on member interactions, resulting in communication delays and breakdowns, in turn affecting task outcomes [35]. Therefore, we posit:

*H1. More even distribution of team members across team boundaries is associated with lower coordination effectiveness in global software development.*

### 2.3 Effect of User Requirements Dynamism on Coordination Effectiveness

Prior research in requirements engineering has reported on the inherent difficulties in defining user requirements due to z communication gap between different groups of stakeholders, and this problem is exacerbated when a team is globally distributed and requirements dynamically change [27], [43], [44]. Consistent with prior research [45], we define user requirements dynamism as the rate of change in user requirements for the software under development. User requirements dynamism causes uncertainty, ambiguity, and variability in a software development project [15].

User requirements dynamism makes it important for the software team to develop a good understanding of changing user needs, assess the business impact of these changes, evaluate technical feasibility, negotiate with users, and modify the software system. All these activities require substantial coordination among team members. The impact of user requirements dynamism on software development is expected to be particularly well pronounced in distributed environments due to increased communication barriers imposed by multiple global boundaries. Therefore, we posit:

*H2. User requirements dynamism is negatively associated with coordination effectiveness in global software development.*

### 2.4 Moderating Effects of Process Rigor, Standardization, Agiity, and Customizability

Software process plays an important role in software development. To cope with the challenges resulting from geographic dispersion and volatile user requirements, software teams need strong software process capabilities [24], [27], [46]. Effective software processes enable teams to cope with various development challenges, reduce severe defects, and fulfill user requirements successfully [47], [48]. We argue that globally dispersed teams need ambidextrous process capabilities demonstrating both alignment and adaptability [49], [50], [51]. On the one hand, globally dispersed teams may need alignment and coherence among all the activities in their software process so that they can carry out software development tasks with stability, consistency, accuracy, and efficiency and thus overcome difficulties in communication and coordination across locations and other boundaries [27]. On the other hand, globally dispersed teams may also need process adaptability to deliver software that meets changed user requirements resulting from greater uncertainty and dynamism in distributed environment [52]. Drawing upon prior literature and our preliminary field interviews, this research identifies process rigor and process standardization as two facets of alignment and process agility and process customizability as two facets of adaptability.

Prior research has viewed process rigor as an important process capability for software development [53]. Following prior literature [27], [50], [54], process rigor is defined in this research as the process capability that increases clarity, accuracy, and formality of the software process. It is characterized by clear definitions of roles, activities, work products, methods, and measures, detailed top-down planning, detailed documentation, tight monitoring and controlling, and use of formal methods [27], [54], [55]. It is especially emphasized and enacted by plan-based structured approaches to software development [50].

Rigorous processes might be especially important when software teams are globally dispersed. The dispersion of geographic, time, cultural, and organizational boundaries in a software team increases the complexity of team coordination and exacerbates the risks of errors, delays, and higher costs. Rigorous processes can reduce these risks by eliminating ambiguity and improving clarity in work procedures. With a high level of process rigor, details of software development tasks are clearly defined and accurately executed. Dispersed teams would benefit from such detailed, well-defined processes because rigorous processes help minimize the need for ad hoc coordination and communication for clarifying and resolving issues and problems. A related prior study demonstrates that the benefit of improved software processes is more salient when the complexity of software development is higher [47]. Therefore, we posit:

*H3. As software process rigor increases, the negative effect of global team dispersion on coordination effectiveness decreases.*

Process standardization has been considered another important process capability to improve software development performance [56]. Process standardization and process rigor are two different, distinct capabilities. While process rigor relates to clarity, accuracy, and reliability of software processes, process standardization mainly refers to uniform and consistent use of the same software processes within the global software team. While some prior studies conceptualized process standardization as process consistency across projects in the organization [56], following prior research on distributed software development [57] this research conceptualizes process standardization as the consistent use of the same methodologies, tools, techniques, templates, and work practices across global development sites.

Communicating spontaneously, frequently, and unambiguously is not done easily in globally dispersed environments because team boundaries need to be bridged and crossed [46]. Communication between globally distributed developers can be problematic because of differences in software methodologies, national and organizational culture, and language barriers. It is very challenging for global teams to develop shared understanding because members of such teams often do not stand on "common ground" [58]. Standardization has been long viewed as an important organizational mechanism to cope with task dependencies [32]. Standardized processes across project locations help overcome communication barriers and facilitate coordination among global team members who otherwise might

have very different ways of carrying out tasks [57]. Standardized processes help team members develop shared mental models about the task [59], set common expectations, understand each other's work, coordinate their activities, and thus integrate distributed work effectively [27], [56]. Process standardization leads to a cohesive organizational culture through the common technical language, procedures, and goals [60]. Furthermore, process standardization can reduce the negative impact of personnel turnover and help overcome differences derived from diverse local contexts [56], [57], [60]. Therefore, process standardization can save a great deal of time and effort because, without standardized processes in place, global software teams would need to spend a significant amount of time and effort to continuously negotiate ground rules and protocols for task coordination. Therefore, we posit:

*H4. As software process standardization increases, the negative effect of global team dispersion on coordination effectiveness decreases.*

Consistent with prior research, process agility in this research is defined as the process capability to sense and respond to changing user requirements [61], [62], [63], [64]. The benefits of process agility have been demonstrated mostly for collocated development, and the generalizability of such findings to globally dispersed development has not yet been validated [64], [65], [66]. As a result, the role of process agility in global software development is poorly understood [52], [67].

Process agility enables software teams to manage their responses to changing user requirements and implement necessary changes in a timely and cost-effective manner [68], [69]. Due to inherent uncertainty in business and technology environments, changes in user requirements are inevitable. Process agility is needed especially when required changes are not anticipated or adequately specified at the onset of the project [70]. Changes in user requirements may disrupt the rhythm and pace of the team's coordination and cause time/cost overruns if the team's response is not timely. Agile software processes enable the team to effectively manage team interactions and collaboration to orchestrate interdependent tasks that need to be executed in multiple locations across boundaries. Thus, the negative effect of user requirements dynamism on software development is reduced by process agility. Therefore, we posit:

*H5. As software process agility increases, the negative effect of user requirements dynamism on coordination effectiveness decreases.*

Process customizability is conceptually distinct from process agility. Whereas process agility refers to the ability to respond to user requirement changes within the already established process, process customizability refers to the ability to respond to user requirement changes by tailoring, reconfiguring, and improvising the process itself [71], [72]. Customizability captures "structural" adaptations of software processes, while agility does not imply such structural changes but rather captures "nonstructural" improvements on the existing processes. In a sense, customizability can be viewed as a metaroutine that changes first-order routines and processes [73]. For example, software teams might increase agility by making a change management process more efficient, having an experienced person dedicated to responding to change requests, using social media and other advanced technology to facilitate communication, and reducing excessive documentation associated with change process. On the other hand, a software team might increase customizability by making processes more modular and loosely coupled, eliminating processes that don't add much value, and improvising new processes to handle exceptions.

Customizing a software process is not uncommon in practice. Prior research found that globally dispersed teams often required the use of a modified method or changes to the extant method [74]. Development methods may be formalized in an organization as a starting base or a process template, but teams customize and reconfigure these methods to accommodate the specific, idiosyncratic context of a given project [71], [72], [75], [76]. Process customization may change the formality, frequency, granularity, and scope of process activities, documents, artifacts, roles, and phases [77].

In dynamic business and technology environments, it is difficult for a software team to determine the most suitable and effective development process at the outset of the project. Significant changes in business and technology would demand the customization of software processes throughout the software development life cycle as existing processes may no longer be best suited to changed user requirements and development environments [77]. Therefore, we argue that process customization is an important way to deal with changing user requirements and a lack of such customizability can be a barrier to coordination of team members. It is critical for software teams to be able to easily tailor and reconfigure the extant processes, thus dynamically adapting and optimizing software processes to effectively handle changing user requirements and mitigate the negative impact of the changes on team coordination effectiveness [46]. Therefore, we posit:

*H6. As software process customizability increases, the negative effect of user requirements dynamism on coordination effectiveness decreases.*

### 2.5 Effect of Coordination Effectiveness on Global Software Development Success

Because global software development is a complex task with highly dependent activities, we argue that project coordination effectiveness is essential to the success of the software development project. The positive effects of team coordination effectiveness on project performance have been proposed in the software engineering literature [4], [18]. Coordination theory predicts that coordination has an impact on task performance when tasks have highly dependent activities [35], [78], [79], as is the case with software development work [80]. The difficulties and problems associated with team coordination is one of the main reasons for time and cost overruns, poor quality, and other problems associated with software development [81]. Effective coordination is likely to help the software development project to meet time, cost, and scope goals and deliver a high-quality system that would satisfy users and benefit the implementing organization [82].

Prior research suggests that project success in software development consists of two different dimensions: process

performance and product performance [48], [79]. Process performance refers to the effectiveness of the implementation process undertaken, for example, in terms of on-time completion and within-budget completion of the project [79], [83]. In the context of global software development, another important indicator of process performance would be how well the project team leveraged distributed resources to materialize the potential advantage of being globally dispersed. Product performance refers to the performance of the resulting software system, including software quality, software functionality, software impact, and user satisfaction with the software system [84]. In this study, we use perceived measures for the two dimensions of global software development success: project manager's perception of *software process success* and project stakeholder's perception of *software success*. Therefore, we posit:

*H7. Coordination effectiveness is positively associated with software process success as perceived by the project manager.*

*H8. Coordination effectiveness is positively associated with the success of the resulting software as perceived by the system's stakeholders.*

In addition to the hypotheses we posited above, we expect the positive effects of process rigor, process standardization, process agility, and process customizability on coordination effectiveness. However, since these effects are not the main focus for this research, we did not propose hypotheses explicitly for those effects. Furthermore, we did not hypothesize moderating effects of process rigor and process standardization on the relationship between user requirements dynamism and coordination effectiveness, or moderating effects of process agility and process customizability on the relationship between global team dispersion and coordination effectiveness because we did not find compelling theoretical underpinnings that support such moderating effects.

## 3 Method

### 3.1 Data Collection and Research Sample

Before collecting primary data with our online survey instrument, we conducted preliminary field interviews with several project managers of global software development projects to formulate research questions, identify key constructs, and generate measurement items. We conducted 1 hour, semi-structured interviews face-to-face and by telephone. We then used a web-based online survey instrument to collect our primary data. No one from the preliminary field interviews participated in the survey. To avoid common method bias, we designed the survey instrument to collect data from two key informants from each project, a project manager and a stakeholder (e.g., a project sponsor, user, or client of the project). Project managers responded to questions related to team boundaries, user requirements dynamism, rigor/consistency/agility/customizability of their software process, coordination effectiveness, and software process success (see Appendix A). Project stakeholders responded to questions related to software success (see Appendix B).

We solicited survey participation from organizations that were members of the Center for IT and the Global Economy at the Kogod School of Business, American University. We identified and sent an invitation to 171 project managers who had managed global software development projects. In total, we received 103 project manager responses. Several responses had missing values for important items such as team distribution. After eliminating those incomplete responses and several redundant responses due to multiple data entries, we retained 80 usable project managers' responses for our data analysis, resulting in an effective response rate of 46.8 percent. A substantial number of our data points came from three US companies: an oil company, a manufacturing company, and an IT service company. Specifically, 67 projects were from these three companies and 13 projects were from 12 different organizations in various industries. We then contacted stakeholders of the projects in our sample whose name and contact information were provided by the project managers. We obtained 66 responses from stakeholders. After eliminating incomplete responses, we retained 62 usable stakeholders' responses. In sum, we retained 62 matched responses from project managers and stakeholders and 18 additional project manager responses that were not matched with their corresponding stakeholder responses. Since the sample size is not very large, we used all 80 project manager responses when testing H1 to H7 that did not require stakeholder data, to maximize the statistical power of our data analysis. However, when testing H8, which required data both from project managers and from stakeholders, we used the 62 matched pairs for data analysis.

On average, a project team had 48.7 members, a budget of $8.3 million, and duration of 15.8 months. Project managers had about 11 years of project management experience and nearly 20 years of IT-related work experience. Table 1 shows further details. To examine the possibility of nonresponse bias, we compared the response group with the nonresponse group on organizational size and industry and did not find significant differences. In addition, we split the sample into two half-sized subgroups based on the time when each response was received. We then compared the early response group with the later response group on variables such as project budget, team size, project type, organizational size, and project management experience, gender, and education. No significant differences between the two groups on these variables were found. Therefore, we concluded that nonresponse bias was not likely to be an issue.

### 3.2 Constructs and Measures

Appendix A shows the measurement items used in this research. We measured global team dispersion by assessing the extent to which the members of a software development team are evenly distributed across boundaries of space, time, culture, and organization. Project managers were asked to identity all locations of the development sites, the number of team members at each site, the number of team members with specific nationalities (nationality is used as a proxy for culture), and the number of members from different organizations such as external vendors and contractors. Then, the number of time zones was computed from project locations. We used the Gini coefficient [85],

TABLE 1
Sample Characteristics

| Variable | Percent | Variable | Percent | Variable | Percent |
|---|---|---|---|---|---|
| **Project profile (n = 80)** | | Number of nationalities | | **Stakeholder profile (n = 62)** | |
| Type of project | | 1 | 19.4 | Education | |
| New development | 27.4 | 2 | 19.4 | College degree | 42.4 |
| Off-the-shelf software | 31.5 | 3 – 5 | 33.3 | Graduate degree | 50.0 |
| Software enhancement | 41.1 | 6 – 8 | 16.1 | Other | 7.6 |
| Team size (mean: 48.7) | | 9 - 20 | 11.8 | Role | |
| Less than 20 members | 38.9 | Number of organizations | | Project sponsor | 24.2 |
| 20 – 50 members | 34.4 | 1 | 21.5 | User | 10.6 |
| Over 50 members | 26.7 | 2 | 22.6 | Other stakeholders | 65.2 |
| Budget (mean: $8.3 million) | | 3 – 5 | 38.7 | Rank/job title | |
| Less than $1 million | 32.4 | 6 – 11 | 12.2 | Manager | 74.2 |
| $1 – 5 million | 32.4 | User distribution | | Executive | 13.6 |
| Over $5 million | 35.2 | Collocated | 8.9 | Specialist | 12.2 |
| Duration (mean: 15.8) | | Domestically dist. | 17.8 | | |
| Less than 6 months | 12.4 | Globally distributed | 73.3 | **Organization profile** | |
| 6-11 months | 31.4 | | | Industry | |
| 12 – 23 months | 31.5 | **Project manager profile (n = 80)** | | Manufacturing | 16.3 |
| Over 24 months | 24.7 | Education | | Software/IT service | 31.5 |
| Development methodologies | | College degree | 48.4 | Utility/commodity | 40.2 |
| Waterfall/structural | 45.2 | Graduate degree | 42.0 | Non-for-profit | 3.3 |
| Agile/iterative | 26.1 | Other | 9.6 | Bank/finance | 2.2 |
| Hybrid/custom | 28.7 | Gender | | Other | 6.5 |
| Number of geographic locations | | Female | 21.5 | Employees (mean: 102,999) | |
| 1 – 3 | 41.9 | Male | 78.5 | Less than 10,000 | 17.8 |
| 4 – 6 | 26.9 | Project management experience (mean 11.1) | | 10,000 – 100,000 | 40.0 |
| 7 – 9 | 14.0 | Less than 10 years | 40.0 | Over 100,000 | 42.2 |
| 10 – 25 | 17.1 | 10 – 20 years | 53.5 | | |
| Number of time zones | | Over 20 years | 6.5 | | |
| 1 | 17.2 | IT work experience (mean 19.9) | | | |
| 2 | 26.9 | Less than 10 years | 10.8 | | |
| 3 – 4 | 23.7 | 10 – 20 years | 46.2 | | |
| 5 – 6 | 19.4 | Over 20 years | 43.0 | | |
| 7 - 10 | 12.8 | | | | |

which is a widely used measure of dispersion, to measure how evenly a project team was dispersed across multiple team boundaries. We computed the Gini coefficient for each of the four team boundaries and then averaged the four Gini coefficients to indicate a composite scale of global team dispersion. The resulting mean Gini coefficient ranges from 0 (i.e., team members are perfectly evenly dispersed across different locations, time zones, nationalities, and organizations) to 1 (i.e., all team members are concentrated in one location, time zone, nationality, and organization). We used the reversed mean Gini coefficient (i.e., [*1-mean Gini coefficient*]) as the final measure of global team dispersion to indicate how evenly teams are distributed across global team boundaries so that higher mean scores indicate more even distribution of team members.

The measurement items for user requirements dynamism were informed by and adapted from prior research [15], [86]. User requirements include both functional and nonfunctional requirements [81], [87]. Using five-point Likert scale items, we measured rate of changes in 10 different categories of requirements, including system input, output, data structure, data processing logic, business process, user interface, software interface with other software systems, security, reliability, and data backup/recovery. In addition, two items measured the extent of requirements fluctuation during software development.

We measured process rigor using items adapted from the prior literature (e.g., Jalote [54] and Ahern et al. [81]) and from results of our preliminary field interviews. The items intend to measure the extent to which software process is defined, formalized, planned, communicated, monitored, and measured. The items for process standardization measure the extent to which common development processes are consistently used across project sites. Specifically, we measured the consistent use of software development methodologies, project management processes, communication methods/technologies, and project performance review methods/processes across sites. These items were selected based mainly on the results of our preliminary field interviews because no directly relevant measures of process

TABLE 2
Means, Standard Deviations, Scale Reliability, and Intercorrelations

| Variable | Mean | SD | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. *ln* Team Size | 3.36 | 1.22 | – | | | | | | | | | | | |
| 2. *ln* Project Duration | 2.61 | 0.68 | 0.43** | – | | | | | | | | | | |
| 3. *ln* PM Experience | 2.16 | 0.75 | 0.24* | 0.17 | – | | | | | | | | | |
| 4. Process Rigor | 3.80 | 0.66 | 0.08 | 0.05 | 0.19 | (0.84) | | | | | | | | |
| 5. Process Standardization | 3.99 | 0.70 | 0.04 | 0.09 | 0.01 | 0.56** | (0.85) | | | | | | | |
| 6. Process Agility | 3.75 | 0.68 | 0.07 | 0.07 | 0.08 | 0.52** | 0.49** | (0.84) | | | | | | |
| 7. Process Customizability | 3.87 | 0.62 | 0.17 | 0.24* | 0.19 | 0.51** | 0.38** | 0.38** | (0.86) | | | | | |
| 8. Team Dispersion | 0.59 | 0.18 | -0.26* | -0.09 | -0.05 | 0.15 | 0.22 | 0.23* | 0.01 | – | | | | |
| 9. Requirements Dynamism | 2.48 | 0.72 | -0.01 | -0.11 | -0.16 | -0.26* | -0.32** | -0.20 | -0.01 | 0.03 | – | | | |
| 10. Coordination Effectiveness | 3.80 | 0.70 | 0.01 | 0.21 | 0.26* | 0.57** | 0.60** | 0.62** | 0.36** | -0.02 | -0.46** | (0.85) | | |
| 11. Software Process Success | 3.52 | 0.78 | -0.13 | 0.01 | 0.14 | 0.61** | 0.35** | 0.34** | 0.23* | 0.07 | -0.21 | 0.59** | – | |
| 12. Software Success | 3.85 | 0.60 | -0.22 | -0.01 | 0.16 | 0.52** | 0.21 | 0.18 | 0.06 | 0.06 | -0.40** | 0.58** | 0.45** | (0.89) |

*Notes. 1) * 0.05 level; ** 0.01 level, 2) Cronbach's alphas reported on the diagonal, 3)* $n = 80$ *except for Software Success (*$n_{softwaresuccess} = 62$*).*

standardization for global software development were available from prior research. We measured process agility by assessing how software process enables the team to sense requirements changes, plan for appropriate responses, and incorporate necessary changes into the software under development. These measurement items were informed by and adapted from prior research [62], [63]. We measured process customizability by assessing how effectively the team was able to tailor and reconfigure extant software processes and design and implement new processes when needed. The items were selected based on the results of our field interviews.

Coordination effectiveness is defined as the extent to which task dependencies have been effectively managed as evidenced by reduced coordination problems [17]. Software development projects are considered to be poorly coordinated when they exhibit task redundancy, rework, lack of problem-solving capability, and integration-related difficulties. The items were adapted from similar scales used in prior research [40] to measure coordination problems. For our data analysis, the item scores measuring coordination problems were reversed to indicate coordination effectiveness. The success of global software development was measured by two dimensions [79], [88]. The first dimension is the process aspect of the software development performance perceived by project managers. We measured time and cost overruns, the extent to which the team saved time and cost in globally dispersed development environment, and the extent to which the team effectively utilized distributed resources and talents. The second dimension is the success of the completed software system as perceived by project stakeholders [84], [89]. The items measured the extent to which the software met user requirements, functional goals and technical specifications, and the extent to which the software provided useful information, satisfied users, produced expected organizational benefits, and was easy to use.

# 4 Analysis and Results

## 4.1 Construct Validity

In this research, the measurement items of process rigor, process standardization, process agility, process customizability, coordination effectiveness, and software success are intended to be reflective measures, whereas the measurement items of global team dispersion, user requirements dynamism, and process success are intended to be formative measures that tap into the multifacets of the constructs.[1] Following recommendations from prior research [90], we used different approaches to validate the convergent validity, discriminant validity, and reliability of reflective measures versus formative measures. Exploratory factor analysis of the reflective measures produced the hypothesized six-factor solution (see Appendix C). All retained items loaded on their expected factors with loadings ranging from 0.60 to 0.90. Results show that the square root values of the average variance extracted (AVE) for all six constructs are greater than 0.707 and exceed the correlations with other constructs, indicating satisfactory convergent and discriminant validities for the constructs. Cronbach's $\alpha$ values for the six reflective constructs ranged from 0.84 to 0.89, indicating adequate construct reliabilities (Table 2).

We followed the guidelines recommended by Petter et al. [90] to validate the construct validity of the formative measures. We used a modified multitrait-multimethod matrix (MTMM) analysis proposed by Loch et al. [91] for validating convergent and discriminant validity of formative measures. Results show that the majority of

1. For rigorous validation of measures, researchers have recently begun to distinguish formative measures from reflective measures (For more information, see [90]). Reflective measures are caused by their latent construct and thus they are expected to have high degrees of covariation among them. On the other hand, formative measures are causes of their latent construct and thus they are not expected to have high degrees of covariation among them. Reflective measures and formative measures require different approaches and criteria for validating reliability, convergent validity, and discriminant validity.

interitem correlations and item-to-construct correlations within a given construct are significant and that most items correlate more highly with one another within the same construct than with items of other constructs.[2] Therefore, convergent and discriminant validities for the three formative constructs were satisfactory.

To assess if significant common method variance is present [92], we performed the following analyses. First, we conducted a Harman's one-factor test [93] on all the variables of process rigor, process standardization, process customizability, process agility, user requirements dynamism, coordination effectiveness, and process success. All these variables were entered into an exploratory factor analysis. Our results show that multiple factors with eigenvalue greater than 1 are present and that the largest factor does not account for a majority of the covariance, accounting for only 27.07 percent. Second, we loaded all these variables onto one factor and examined the fit of the confirmatory factor analysis model. If a substantial amount of common method variance is present, the one-factor confirmatory factor analysis model should fit the data well [94]. Our results do not show a good fit of the single-factor model with the data ($\chi_2(\text{d.f.} = 819,\ N = 80) = 2{,}085.81$, $p = 0.000$, $\text{GFI} = 0.44$; $\text{AGFI} = 0.39$; $\text{CFI} = 0.78$; $\text{RMR} = 0.12$; $\text{RMSEA} = 0.14$). The results obtained from the two analyses indicate that common method variance is not of a serious concern and, therefore, is unlikely to confound the interpretation of results.

### 4.2 Model Specification and Estimation Procedure

We specify three equations to test our hypotheses. First, we specify (1) to test H1 to H6. Since this equation includes interaction terms, we centered independent variables before creating interaction terms to avoid possible multicollinearity and lack of scale invariance problems [95]. We controlled for variables that may affect team coordination such as team size, project duration, and project manager's project management experience when we tested the effects of the independent variables and their interaction terms with moderating variables on the dependent variables. We also controlled for software development methodology using dummy variables but dropped them in the final analysis to preserve degrees of freedom because no significant effect was found. We measured team size by the number of team members. Project manager's project management experience was measured in years and project duration was measured in months. Our analysis indicates that these three control variables are distributed in a nonnormal fashion. Therefore, we transformed the variables using a logarithmic transformation to reduce skewness and approximate normality. These transformed data demonstrated normality in Q-Q plots. We included binary dummy variables to control for the three organizations (Organizations 1, 2, and 3) that accounted for a large portion of the sample.

2. These results are available upon request.

$$\begin{aligned}\text{Coordination Effectiveness} &= \alpha_0 + \alpha_1 \ln(\text{Team Size}) \\ &+ \alpha_2 \ln(\text{Project Duration}) + \alpha_3 \ln(\text{PM Experience}) \\ &+ \alpha_4(\text{Org1}) + \alpha_5(\text{Org2}) + \alpha_6(\text{Org3}) + \alpha_7(\text{Team Dispersion}) \\ &+ \alpha_8(\text{RequirementsDynamism}) + \alpha_9(\text{Process Rigor}) \\ &+ \alpha_{10}(\text{Process Standardization}) + \alpha_{11}(\text{Process Agility}) \\ &+ \alpha_{12}(\text{Process Customizability}) \\ &+ \alpha_{13}(\text{Team Dispersion} \times \text{ProcessRigor}) \\ &+ \alpha_{14}(\text{Team Dispersion} \times \text{Process Standardization}) \\ &+ \alpha_{15}(\text{Requirements Dynamism} \times \text{Process Agility}) \\ &+ \alpha_{16}(\text{Requirements Dynamism} \times \text{Process Customizability}) \\ &+ \varepsilon_1. \end{aligned} \tag{1}$$

We specify (2) to test H7. Although not hypothesized in this research, the equation also included the main effects of team boundary dispersion, user requirements dynamism, process rigor, process standardization, process agility, and process customizability on software process success to examine whether or not these variables have direct main effects on the dependent variable.

$$\begin{aligned}\text{Software Process Success} &= \beta_0 + \beta_1 \ln(\text{TeamSize}) \\ &+ \beta_2 \ln(\text{Project Duration}) + \beta_3 \ln(\text{PM Experience}) \\ &+ \beta_4(\text{Org 1}) + \beta_5(\text{Org 2}) + \beta_6(\text{Org 3}) \\ &+ \beta_7(\text{Coordination Effectiveness}) \\ &+ \beta_8(\text{Team Dispersion}) + \beta_9(\text{Requirements Dynamism}) \\ &+ \beta_{10}(\text{Process Rigor}) + \beta_{11}(\text{Process Standardization}) \\ &+ \beta_{12}(\text{Process Agility}) + \beta_{13}(\text{Process Customizability}) \\ &+ \varepsilon_2. \end{aligned} \tag{2}$$

In a similar fashion, we specify (3) to test H8.

$$\begin{aligned}\text{Software Success} &= \gamma_0 + \gamma_1 \ln(\text{Team Size}) \\ &+ \gamma_2 \ln(\text{Project Duration}) + \gamma_3 \ln(\text{PM Experience}) + \gamma_4(\text{Org 1}) \\ &+ \gamma_5(\text{Org 2}) + \gamma_6(\text{Org 3}) + \gamma_7(\text{Coordination Effectiveness}) \\ &+ \gamma_8(\text{Team Dispersion}) + \gamma_9(\text{Requirements Dynamism}) \\ &+ \gamma_{10}(\text{Process Rigor}) + \gamma_{11}(\text{Process Standardization}) \\ &+ \gamma_{12}(\text{Process Agility}) + \gamma_{13}(\text{Process Customizability}) + \varepsilon_3. \end{aligned} \tag{3}$$

### 4.3 Results

We estimated the parameters of the three equations using a hierarchical ordinary least squares (OLS) regression method. Tables 3, 4, and 5 show the analysis results for (1), (2), and (3), respectively. We conducted Kolmogorov-Smirnov tests to check the normality of the residuals and did not find any violations ($z = 0.61, p = 0.86$ for (1); $z = 0.54, p = 0.93$ for (2); $z = 0.42, p = 0.98$ for (3)). We tested for heteroscedasticity of the error terms using White's tests and did not find any violations.

For (1), the predictive power of the regression model increased significantly as we added to the base model (Model 1a) the main effects (Model 1b; $\Delta\text{F} = 26.696, p < 0.01$) and the hypothesized interaction effects (Model 1c; $\Delta F = 3.166, p < 0.05$). The estimates of the parameters

TABLE 3
Results of Regression Analysis for Coordination Effectiveness (1)

| Variables | Model 1a Coeffi. (Std. error) | Model 1b Coeffi. (Std. error) | Model 1c Coeffi. (Std. error) |
|---|---|---|---|
| Intercept ($\alpha_0$) | 3.214** (0.385) | 3.556** (0.259) | 3.437** (0.257) |
| *ln* Team Size ($\alpha_1$) | - 0.072 (0.068) | - 0.143** (0.046) | - 0.124** (0.046) |
| *ln* Project Duration ($\alpha_2$) | 0.072 (0.128) | 0.174* (0.086) | 0.160† (0.081) |
| *ln* PM Experience ($\alpha_3$) | 0.265* (0.103) | 0.133† (0.072) | 0.114 (0.073) |
| Org 1 ($\alpha_4$) | - 0.305 (0.227) | - 0.107 (0.152) | 0.078 (0.157) |
| Org 2 ($\alpha_5$) | 0.311 (0.209) | - 0.065 (0.144) | 0.066 (0.155) |
| Org 3 ($\alpha_6$) | 0.119 (0.333) | 0.315 (0.224) | 0.435† (0.219) |
| H1: Team Dispersion ($\alpha_7$) | | -- 0.950** (0.289) | - 0.848** (0.287) |
| H2: Requirements Dynamism ($\alpha_8$) | | -- 0.193** (0.073) | - 0.217** (0.069) |
| Process Rigor ($\alpha_9$) | | 0.186† (0.100) | 0.295** (0.107) |
| Process Standardization ($\alpha_{10}$) | | 0.276** (0.092) | 0.194* (0.093) |
| Process Agility ($\alpha_{11}$) | | 0.459** (0.089) | 0.414** (0.090) |
| Process Customizability ($\alpha_{12}$) | | - 0.065 (0.096) | 0.049 (0.097) |
| H3: Team Dispersion x Process Rigor ($\alpha_{13}$) | | | - 0.439 (0.661) |
| H4: Team Dispersion x Process Standardization ($\alpha_{14}$) | | | 1.259* (0.488) |
| H5: Requirements Dynamism x Process Agility ($\alpha_{15}$) | | | 0.251* (0.101) |
| H6: Requirements Dynamism x Process Customizability ($\alpha_{16}$) | | | - 0.151 (0.117) |
| *F-Statistic* | 3.689** | 13.454** | 12.187** |
| Degrees of Freedom | 6, 73 | 12, 67 | 16, 63 |
| $\Delta F$ | | 26.696** | 3.166* |
| $R^2$ | 0.233 | 0.707 | 0.756 |
| $\Delta R^2$ | | 0.474 | 0.049 |
| Adjusted $R^2$ | 0.170 | 0.654 | 0.694 |

*Notes. 1)* † $p < 0.10$; * $p < 0.05$; ** $p < 0.01$. *2)* $n = 80$.

TABLE 4
Results of Regression Analysis for Software Process Success (2)

| Variables | Model 2a Coefficient (Std. error) | Model 2b Coefficient (Std. error) |
|---|---|---|
| Intercept ($\beta_0$) | 1.937** (0.462) | 0.841 (0.702) |
| *ln* Team Size ($\beta_1$) | - 0.129* (0.063) | - 0.137* (0.064) |
| *ln* Project Duration ($\beta_2$) | - 0.025 (0.113) | 0.027 (0.114) |
| *ln* PM Experience ($\beta_3$) | 0.098 (0.099) | 0.075 (0.099) |
| Org 1 ($\beta_4$) | - 0.222 (0.201) | - 0.124 (0.199) |
| Org 2 ($\beta_5$) | - 0.244 (0.194) | - 0.204 (0.200) |
| Org 3 ($\beta_6$) | - 0.323 (0.280) | - 0.275 (0.276) |
| H7: Coordination Effectiveness ($\beta_7$) | 0.542** (0.095) | 0.399** (0.128) |
| Team Dispersion ($\beta_8$) | | 0.073 (0.413) |
| Requirements Dynamism ($\beta_9$) | | 0.098 (0.099) |
| Process Rigor ($\beta_{10}$) | | 0.444** (0.130) |
| Process Standardization ($\beta_{11}$) | | - 0.021 (0.126) |
| Process Agility ($\beta_{12}$) | | - 0.077 (0.119) |
| Process Customizability ($\beta_{13}$) | | - 0.016 (0.118) |
| *F-Statistic* | 7.413** | 5.606** |
| Degrees of Freedom | 7, 72 | 13, 66 |
| $\Delta F$ | | 2.453* |
| $R^2$ | 0.419 | 0.431 |
| $\Delta R^2$ | | 0.106 |
| Adjusted $R^2$ | 0.362 | 0.525 |

*Notes. 1)* † $p < 0.10$; * $p < 0.05$; ** $p < 0.01$. *2)* $n = 80$.

TABLE 5
Results of Regression Analysis for Software Success (3)

| Variables | Model 3a Coefficient (Std. error) | Model 3b Coefficient (Std. error) |
|---|---|---|
| Intercept ($\gamma_0$) | 2.303** (0.452) | 2.411** (0.703) |
| *ln* Team Size ($\gamma_1$) | - 0.062 (0.066) | - 0.049 (0.068) |
| *ln* Project Duration ($\gamma_2$) | - 0.026 (0.105) | 0.024 (0.109) |
| *ln* PM Experience ($\gamma_3$) | 0.098 (0.118) | 0.071 (0.123) |
| Org 1 ($\gamma_4$) | 0.105 (0.202) | 0.248 (0.206) |
| Org 2 ($\gamma_5$) | 0.284 (0.185) | 0.348† (0.195) |
| Org 3 ($\gamma_6$) | - 0.323 (0.241) | - 0.224 (0.251) |
| H8: Coordination Effectiveness ($\gamma_7$) | 0.390** (0.095) | 0.274* (0.127) |
| Team Dispersion ($\gamma_8$) | | 0.411 (0.384) |
| Requirements Dynamism ($\gamma_9$) | | - 0.083 (0.109) |
| Process Rigor ($\gamma_{10}$) | | 0.329* (0.125) |
| Process Standardization ($\gamma_{11}$) | | - 0.152 (0.120) |
| Process Agility ($\gamma_{12}$) | | - 0.138 (0.109) |
| Process Customizability ($\gamma_{13}$) | | 0.002 (0.117) |
| *F-Statistic* | 6.691** | 4.446** |
| Degrees of Freedom | 7, 54 | 13, 48 |
| $\Delta F$ | | 1.443 |
| $R^2$ | 0.464 | 0.546 |
| $\Delta R^2$ | | 0.082 |
| Adjusted $R^2$ | 0.395 | 0.423 |

*Notes. 1)* † $p < 0.10$; * $p < 0.05$; ** $p < 0.01$. *2)* $n = 62$.

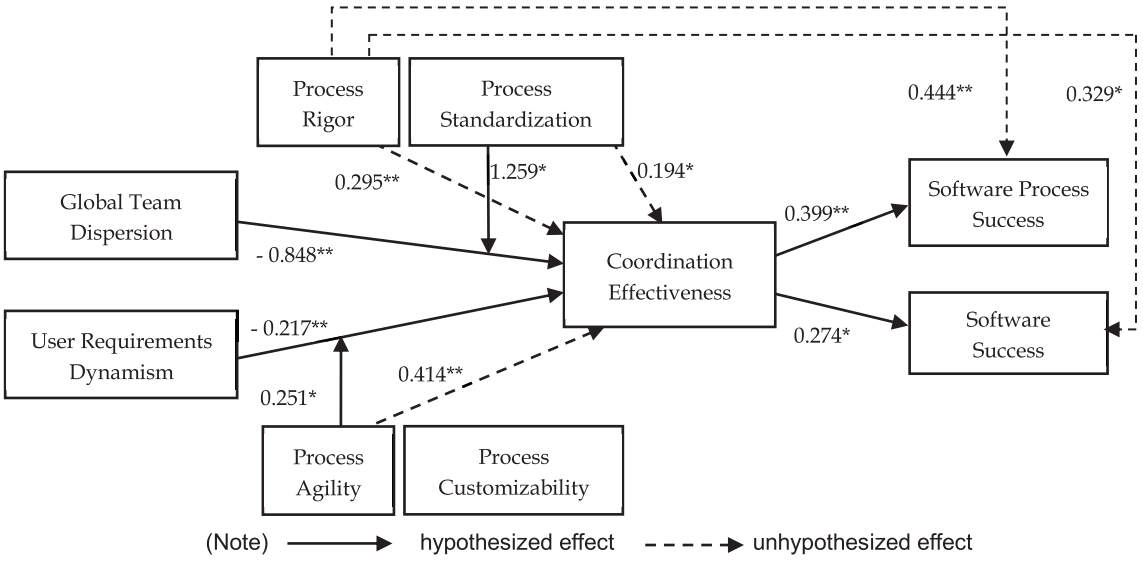



Fig. 2. Summary of regression results.

shown in Models 1a to 1c provide support for H1, H2, H4, and H5: Both global team dispersion across boundaries and user requirements dynamism are associated with lower coordination effectiveness; software process standardization mitigates the negative effect of global team dispersion on coordination effectiveness; and software process agility mitigates the negative effect of user requirements dynamism on coordination effectiveness. However, H3 and H6 were not supported. Although not explicitly hypothesized in this research, we also found positive main effects of process rigor, process standardization, and process agility on coordination effectiveness. However, we found that the main effect of process customizability on coordination effectiveness was not significant. *In sum, these results suggest that global team dispersion and user requirements dynamism have negative main effects on coordination effectiveness and that these negative effects are mitigated by process standardization and process agility, respectively.*

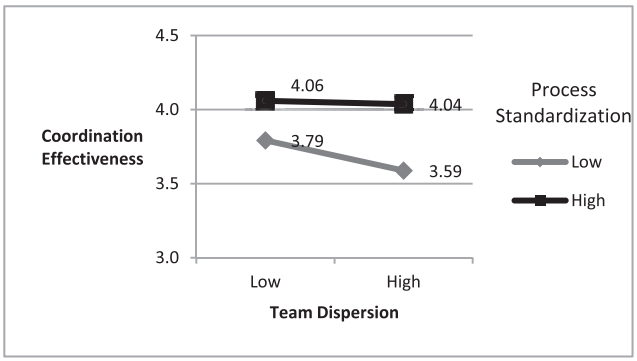
The results for (2) shown in Models 2a and 2b provide strong support for the positive main effect of coordination effectiveness on software process success. The predictive power of the regression model increased significantly as we added to the base model the main effects of the six independent variables ($\Delta F = 2.453, p < 0.05$). The results in Model 2b show that the effect of coordination success on process success remains significant even when the other independent variables are added to the model. Interestingly, we found that process rigor has a positive effect on process success above and beyond its indirect effect mediated by coordination effectiveness.

The results for (3) shown in Models 3a and 3b provide strong support for the positive main effect of coordination effectiveness on software success. The predictive power of the regression model did not increase significantly as we added to the base model the main effects of the six independent variables ($\Delta F = 1.443, p > 0.10$). The results in Model 3b show that the effect of coordination success on software success remains significant even when the other independent variables are added to the model, and that process rigor once again has a positive effect on software success above and beyond the mediating effect of coordination effectiveness. In sum, the estimates of (2) and (3) provide support for H7 and H8 that coordination effectiveness is positively associated with both software process success and software success. Fig. 2 presents the resulting research model based on the empirical tests of hypotheses H1 through H8.

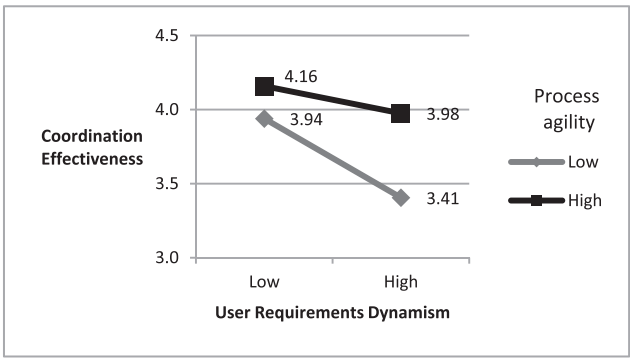
Fig. 3 illustrates the interaction effect of global team dispersion and process standardization and the interaction effect of user requirements dynamism and process agility on coordination effectiveness.[3] When process standardization is low, a relatively strong negative effect of team dispersion on coordination effectiveness is observed, demonstrating a declining line. When process standardization is high, however, this negative effect becomes much weaker, demonstrating a much flatter line (Fig. 3a). Similarly, when process agility is low, a strong negative effect of requirements dynamism on coordination effectiveness is observed, demonstrating a steep line. When process agility is high, however, this negative effect becomes weaker, demonstrating a flatter line (Fig. 3b).

We inspected our models for multicollinearity using conditions specified in prior research [96]. The highest variance inflation factor of all the independent variables and the interaction terms in Models 1a to 3b was 3.83. The condition indices for Models 1a, 1b, 1c, 2a, and 3a were below 20. However, the condition indices for Models 2b and 3b were 42.47 and 45.75, respectively. These relatively high-condition indices may result from high correlations between coordination effectiveness and other independent variables as shown in Table 2. To further assess multicollinearity, we tested reduced models only with the variables exhibiting significant effects in Models 2b and 3b. Results showed that these variables remained significant with the same directions in the reduced models and that the condition indices

3. We constructed the diagrams in Fig. 3 by splitting the data at the median of each variable and calculating the mean of coordination effectiveness for each of the four subgroups. Therefore, the diagrams are only simplified illustrations of the interaction effects and should not be interpreted as perfectly accurate representations of the interaction effects.




(a) Interaction effect of team dispersion and process standardization




(b) Interaction effect of user requirements dynamism and process agility

Fig. 3. Illustration of interaction effects.

were 17.37 and 15.53. In sum, the results indicate the absence of serious multicollinearity effects in our regression models and the robustness of the regression estimates.

## 5 Discussion

### 5.1 Effect of Team Dispersion and Requirements Dynamism

Our study shows the negative impact of global team dispersion and user requirements dynamism (task environment complexity) on coordination effectiveness and on global software development success. High global team dispersion not only increases the number of information cues that need to be processed to work together in a task (e.g., who is where? what time is in location x? did colleagues leave for the day already? how do I word this memo so that my colleagues across the Atlantic understand what I mean?), but also the interdependencies among these information cues, both of which make the task environment more complex [9], [12]. Our preliminary field interviews also support the negative effect of global team dispersion on coordination effectiveness. For example, one software development project manager said *"I would think that it would be foolish to not take whatever mathematical calculation you have (for a collocated project) and start off with 1.5 times that budget and plan amount for a distributed project because it's simply much more difficult to get things done."* Another manager stated, *"When your team is distributed, it takes a lot longer time to figure out certain things that would probably get resolved by a 5 minute phone call ... they dwell on the issue or problem for probably a week or two weeks."*

As we expected, frequent changes in user requirements undermine software teams' ability to seamlessly coordinate development tasks that are often interdependent across team boundaries. The need to deliver software on-time and within-budget, and accommodate changing requirements concurrently creates tensions and confusion that might cause redundant work, rework, and integration challenges. One project manager in our field interview highlighted the challenges associated with user requirements dynamism: *"If something changes, she would ask: OK, you just threw all this stuff at me ... what do you want me to work on first because I can't do it all by the time you need it ... what's your priority?"* Prior research has shown similar negative effects of user requirements dynamism on various software development processes. However, this research extends the generalizability of the relationship to globally distributed software development. Global software teams should understand the strong negative effect of user requirements change and develop proper coping strategies to ensure effective team coordination.

### 5.2 Moderating Effect of Process Standardization and Agility

As the team needs to bridge the even dispersion of members across multiple boundaries to get the job done, process standardization becomes paramount to maintaining coordination. Consistently, our preliminary field interview results also suggested that successful teams used standardized development processes, especially when the development work was carried out by teams whose members were evenly distributed across development sites. For example, a project manager said *"We have common processes for development of systems. Common processes are very, very key when you have multiple locations and team members are widely dispersed."* Another project manager said *"I think one of the key things we have done is we have internally developed project management automation processes and tools that we use globally from wherever we are."* However, when team members were largely concentrated in a single location, the need for process standardization was less important. Our results suggest that global software teams should standardize software processes to help overcome difficulties resulting from team dispersion across multiple boundaries. Prior research tends to suggest that the main role of process standardization is to increase consistency and reliability across software development projects in an organization. Our study suggests a complementary view that process standardization plays a role in coping with global team dispersion.

Contrary to our expectation, process rigor did not moderate the negative effect of team dispersion on team coordination. Given the positive main effect of process rigor on coordination, this result suggests that while process rigor generally increases coordination effectiveness, it does not influence coordination effectiveness more or less as team dispersion increases. In other words, process rigor is equally important for software development independent

of the degree of team dispersion. One plausible explanation is that the cost of implementing and enforcing rigorous process in a highly dispersed team may offset the additional benefits of rigorous processes for the team.

We found that the negative effect of user requirements dynamism on coordination effectiveness was moderated by process agility. This suggests that agile development methods, which promote process agility over process rigor, could be potentially effective for dynamic environments. Recognizing the detrimental consequences of the lack of process adaptability in software development, organizations have begun to adopt agile development methods such as Extreme Programming and Scrum to improve agility in responding to constantly changing user requirements [68], [69]. However, it is important to note that agile methods are not necessarily equal to process agility [66]. While prior research tends to investigate specific agile methods or specific agile practices and principles [66], [97], our research investigates agility as a process capability to understand its underlying theoretical relationships with global software development performance. As a result, this research deepens our theoretical understanding of process agility, which is one of the three facets of software development agility, along with resource agility and linkage agility [66]. Furthermore, while prior research tends to model agility as an independent variable and requirements uncertainty as a moderator [97], [98], our study models agility as a moderator and requirements dynamism as an independent variable, thus providing a complementary view on their interaction. Our results suggest that software teams should build process agility to cope with requirement changes and that the benefit of agility is greater when user requirements change faster. One project manager we interviewed stated that the team's ability to sense problems and issues was critical for project performance because it allowed them to manage project-related changes in a timely manner: "*It [our communication channel] was always open for any issue. This was a must and not an option. The real communication between the programmers there and the users here was something that had to be constantly going on in order to respond to requirement changes effectively*."

We did not find any significant moderating effect of process customizability on the relationship between user requirements dynamism and coordination or on the relationship between team dispersion and coordination. In sum, we could not find any significant main or interaction effect of process customizability in this research. Future research needs to investigate the conditions under which process customizability becomes a critical capability in global software development.

Taken together, our results suggest that process standardization and process agility play different roles in contributing to the success of globally dispersed software development in a complex and volatile environment. While process standardization mitigates the risk associated with dispersed teams, process agility mitigates the risk associated with dynamic business and technology environments. Therefore, software development teams should assess their task environment complexity and deploy appropriate process capabilities accordingly.

### 5.3 Other Findings

Although not explicitly hypothesized in this research, we also found main positive effects of process rigor, process standardization, and process agility on coordination effectiveness as we would have expected. These process capabilities enable software team members to coordinate development tasks in a structured and orderly manner and help them incorporate changes into the software under development without creating chaotic situations. However, we did not find the main effect of process customizability on coordination effectiveness. We speculate that perhaps it is too difficult to adapt and reconfigure processes during a global software project, creating too much disturbance and chaos, thereby losing continuity of development process [99]. Although process customizability allows development processes to evolve and be optimized to best suit the specific development context and changing user demands, the negative side effects resulting from reconfiguring existing processes could have canceled out the benefits. Further research is required to investigate this issue more deeply.

Additionally, we found that coordination effectiveness positively affects global software development success. One important aspect of our study is that we measure our dependent variable, software development success, with responses from two different groups: project managers and stakeholders of the project, thus reducing concerns with common method bias. What is interesting is that the impact of coordination effectiveness on software development success is supported by both perspectives, the project manager's data on software process success and the project stakeholder's data on software system success. The results are remarkably consistent for these two measures of software development success, thus providing assurance that our process success measures are unbiased and robust. This is consistent with coordination theory in that, to be effective, technical tasks that are highly interdependent need to be well coordinated. While this finding is not entirely novel [35], we included this hypothesis to further confirm this effect and also to provide a stronger validation for our research model. Nevertheless, we know from seminal studies on team performance that internal team ratings of performance often vary from external perceptions [100], so our results reinforce the importance of coordination effectiveness to software development success.

Interestingly, we found that process rigor has direct effects on global software development above and beyond its indirect effect mediated by coordination effectiveness, both in terms of process success and software success. This result suggests that the positive effect of process rigor on global software development is only partially mediated, rather than fully mediated, by increased coordination effectiveness. One plausible explanation is that process rigor improves software development performance not only because it makes coordination among interdependent work more effective but also because it enhances independent work, which does not require coordination, to be more productive and of higher quality. Therefore, coordination does not fully account for the variance in software development performance caused by process rigor. Future

research needs to further investigate these complex mediating paths by which process rigor affects software development.

### 5.4 Contributions

Our research makes an important theoretical contribution to our understanding of the drivers of success in complex global software development work and offers a perspective not investigated in prior research. Our study shows the negative impact of task environment complexity—i.e., global team dispersion and user requirements dynamism—on coordination effectiveness and on global software development success. While prior research focuses on investigating the effect of software task complexity on coordination, our study extends this line of inquiry into the complexity of the environment in which software development task is situated and enacted.

In this research, global team dispersion measures the extent to which a team is evenly dispersed across multiple boundaries including distance, time, culture, and organization and how global team dispersion hampers coordination effectiveness. This is an important theoretical contribution because prior organizational research has focused on the effects of single boundaries (e.g., distance, time zones) and used limited measures of these boundaries (e.g., number of locations, number of time zones), but our study is the first to propose and validate empirically that global team dispersion across these multiple boundaries generates substantial task environment complexity. Furthermore, modeling only one or two boundaries (e.g., distance and time zones) produces biased estimators due to the omission of other boundaries (e.g., culture and organizational) that may have an effect, but on the other hand, including many boundaries in the model causes multicollinearity with unstable predictors and large standard errors. Our measurement approach eliminates these problems and allows us to model the effect of dispersion across multiple global boundaries.

Appendix D compares the effects of different measures of global team boundaries on coordination effectiveness. The *team dispersion over multiple boundaries* we have used in the present study exhibits the strongest negative effect on coordination effectiveness among all measures. None of the single or multiple boundary counts is significantly associated with coordination effectiveness. Team dispersion over a single boundary also shows a weaker effect on coordination effectiveness, thus validating that our measure of dispersion across multiple global boundaries best captures software development environment complexity resulting from global boundaries. We recommend that future studies use this measure for rigorous empirical work.

While our findings about the main effects of global team dispersion and user requirements dynamism contribute to our understanding of globally dispersed software development, uncovering the interactions of these variables with software process capabilities further enhances our knowledge of the more nuanced dynamics in global software development. Interestingly, our research demonstrates that the negative impact of global team dispersion can be mitigated by process standardization and that the negative impact of user requirements dynamism can be mitigated by process agility. Prior studies have uncovered various effects of global team boundaries, but our study is one of the first to show how such negative effects can be mitigated by management interventions aimed at developing effective process capabilities. From a theoretical standpoint, our research suggests that standardization and agility can be viewed as global software process capabilities that neutralize risks resulting from task environment complexity. Furthermore, prior studies tend to use case study methods to examine process alignment and adaptability [49], [98]. Our study is one of the few field survey studies, testing theory with quantitative data and, thus, complementing prior case studies.

### 5.5 Limitations and Conclusions

Our research is not without limitations. Our study is subject to the inherent limitations of any survey research, although we have taken careful steps to validate our scales and measurement model and to avoid problems of common method variance. Our study is also limited by the cross-sectional nature of our data. It would be useful to conduct longitudinal studies with more than one wave of surveys to better understand the causality of the hypothesized relationships. Longitudinal studies can also allow us to examine how the four different process capabilities play out over different phases of the software development life cycle to mitigate the negative effects of team dispersion and user requirements dynamism on team coordination. Another limitation is that process agility and process customizability could be perceived to be related to each other, although they are intended to be conceptually distinct and measured differently. Further investigation on the conceptual and operational differences between the two constructs is warranted. Finally, although our sample size provides sufficient power ($> 0.8$) to detect Cohen's [101] medium and large effects, it provides limited statistical power to detect small effects. Therefore, it is possible that some of the nonsignificant effects in our results might have been detected by a larger sample size with greater statistical power. Future research should further investigate this issue.

In conclusion, developing a software system with globally dispersed team members in highly complex and volatile environments present daunting challenges and risks to coordinating interdependent tasks that eventually influence software development success. While there might be many other mechanisms to cope with such challenges and risks, our study shows that software process capabilities—especially process standardization, process rigor, and process agility—are one possible answer to help software development teams overcome such challenges and mitigate the risks, thus delivering high quality, effective software systems to users.

## APPENDIX

### A. Measurement Items for the Project Manager Survey

#### *I. Global Team Dispersion*

*Note.* This data is collected via a dynamic web survey instrument. The instrument provides the survey respondent

with the following questions. The data is used to compute Gini coefficients for the four team boundaries based on number of geographic locations, number of time zones, number of nationalities/cultures, and number of organizations, along with number of team members distributed across these boundaries:

1. A list of *locations* (countries for the most part, except for large countries with multiple time zones which will have regions listed). The user will be prompted to check all the *geographic locations* in which they had team members working on the project. The list will have a blank box to add locations that are not in the list. The user will then click submit.
2. This will create a record for each location checked by the participant. The next screen will have these locations listed and the participant will be asked to enter the number of *members in each location*.
3. A list of *countries*. The user will be prompted to check all the *nationalities* represented in the team. The list will have a blank box to add countries that are not in the list. The user will then click submit.
4. This will create a record for each country checked by the participant. The next screen will have these locations listed and the participant will be asked to enter the *number of members for each nationality*.
5. A list with only one record labeled "Main Organization," with a blank box to enter any other organizations participating in the project.
6. This will create a record for each organization involved in the project. The next screen will have these organizations listed and the participant will be asked to select *a role for the organization* (e.g., client, developer or maybe requirements, production, etc.), and to *list the number of team members in each organization*.

### II. User Requirements Dynamism

Please indicate the extent to which you agree or disagree with the following statements (1: Strongly disagree, 2: Somewhat disagree, 3: Neutral, 4: Somewhat agree, 5: Strongly agree):

1. System input requirements changed frequently (Dynamism1).
2. System output requirements changed frequently (Dynamism2).
3. Data structure requirements changed frequently (Dynamism3).
4. Requirements for data processing logic changed frequently (Dynamism4).
5. Requirements for business processes embedded in the system changed frequently (Dynamism5).
6. User interface requirements changed frequently (Dynamism6).
7. Requirements for system interface with other systems changed frequently (Dynamism7).
8. System security requirements changed frequently (Dynamism8).
9. System reliability requirements changed frequently (Dynamism9).
10. Data backup/recovery requirements changed frequently (Dynamism10).
11. System requirements fluctuated quite a bit (Dynamism 11).
12. System requirements identified at the beginning of the project were quite different from those existing at the end (Dynamism12).

### III. Software Process Capabilities

Please indicate the extent to which you agree or disagree with the following statements

(1: Strongly disagree, 2: Somewhat disagree, 3: Neutral, 4: Somewhat agree, 5: Strongly agree)

*Process rigor:*

1. Project team responsibilities were clearly defined and communicated (Rigor1).
2. Project team created a detailed project plan (Rigor2).
3. Formal estimation methods were used for project planning (Rigor3).
4. Project team used a formal software development process (Rigor4).
5. Project performance was accurately tracked against planned targets (Rigor5).
6. Progress toward system quality goals was quantified and managed (Rigor6).

*Process standardization:*

1. Common project management practices were used consistently across sites (Standard1).
2. Common project planning methods/techniques were used consistently across sites (Standard2).
3. Common communication methods/technologies were used consistently across sites (Standard3).
4. Common project performance review methods/processes were used consistently across sites (Standard4).

*Process agility:*

1. Project team was able to sense user requirements changes effectively (Agility1).
2. Project team was able to strategize its response to user requirements changes effectively (Agility2).
3. Project team was able to make effective decisions to cope with user requirements changes (Agility3).
4. Project team was able to incorporate user requirements changes into the system effectively (Agility4).

*Process customizability*:

1. Project team was able to effectively tailor its development processes when needed (Customizability1).
2. Project team was able to effectively implement new development processes when needed (Customizability2).
3. Project team was able to effectively reconfigure its development processes when needed (Customizability3).
4. Project team was able to effectively improvise its development processes when needed (Customizability4).

TABLE 6
Factor Loadings and Cross Loadings for Reflective Constructs

| Scale item | Process Rigor | Process standardi-zation | Process Agility | Process Customi-zability | Coordination Effectiveness | Software Success |
|---|---|---|---|---|---|---|
| Rigor5 | **0.82** | 0.09 | 0.08 | 0.11 | 0.15 | 0.28 |
| Rigor4 | **0.75** | 0.16 | 0.01 | 0.19 | 0.03 | 0.17 |
| Rigor2 | **0.67** | 0.04 | 0.01 | 0.18 | 0.28 | 0.23 |
| Rigor1 | **0.64** | 0.13 | 0.21 | 0.09 | 0.19 | 0.27 |
| Rigor3 | **0.63** | 0.15 | 0.14 | 0.17 | 0.26 | 0.12 |
| Rigor6 | **0.60** | 0.12 | 0.23 | 0.06 | 0.25 | -0.02 |
| Standard2 | 0.11 | **0.85** | 0.15 | 0.20 | 0.17 | 0.03 |
| Standard1 | 0.21 | **0.77** | 0.17 | 0.05 | 0.25 | 0.06 |
| Standard4 | 0.22 | **0.76** | 0.09 | 0.09 | 0.16 | 0.12 |
| Standard3 | -0.05 | **0.62** | 0.21 | 0.27 | 0.15 | 0.06 |
| Agility3 | 0.10 | 0.02 | **0.85** | 0.18 | 0.15 | 0.00 |
| Agility2 | 0.09 | 0.26 | **0.79** | 0.09 | 0.29 | -0.09 |
| Agility4 | 0.26 | 0.05 | **0.76** | 0.24 | 0.21 | 0.14 |
| Agility1 | 0.02 | 0.27 | **0.72** | 0.13 | 0.13 | 0.00 |
| Customizability3 | 0.25 | 0.18 | 0.00 | **0.88** | 0.05 | -0.01 |
| Customizability 2 | 0.10 | 0.13 | 0.10 | **0.84** | 0.07 | -0.10 |
| Customizability1 | 0.26 | 0.03 | 0.28 | **0.73** | 0.02 | -0.05 |
| Customizability4 | 0.00 | 0.13 | 0.21 | **0.70** | 0.07 | 0.05 |
| Coordination6 | 0.19 | 0.09 | 0.13 | 0.07 | **0.82** | 0.14 |
| Coordination4 | 0.05 | -0.03 | 0.25 | 0.13 | **0.78** | 0.08 |
| Coordination5 | 0.01 | 0.19 | 0.09 | 0.09 | **0.77** | 0.10 |
| Coordination3 | 0.13 | 0.13 | 0.07 | 0.20 | **0.72** | 0.22 |
| Coordination1 | 0.31 | 0.24 | 0.15 | -0.16 | **0.70** | 0.19 |
| Coordination2 | 0.22 | 0.25 | 0.17 | -0.13 | **0.68** | 0.11 |
| SoftwareSuccess3 | 0.03 | 0.04 | 0.07 | 0.00 | 0.03 | **0.90** |
| SoftwareSuccess1 | 0.23 | 0.04 | 0.03 | 0.08 | 0.13 | **0.83** |
| SoftwareSuccess4 | 0.22 | 0.08 | -0.10 | -0.05 | 0.12 | **0.82** |
| SoftwareSuccess7 | 0.27 | 0.04 | 0.03 | -0.09 | 0.10 | **0.82** |
| SoftwareSuccess5 | 0.06 | -0.04 | 0.01 | 0.03 | 0.21 | **0.81** |
| SoftwareSuccess8 | 0.16 | 0.04 | 0.08 | -0.01 | 0.08 | **0.79** |
| SoftwareSuccess2 | -0.15 | 0.22 | -0.07 | 0.03 | 0.28 | **0.63** |
| SoftwareSuccess6 | 0.29 | -0.02 | -0.04 | -0.12 | -0.04 | **0.61** |

### *IV. Coordination Effectiveness*

Please indicate the extent to which you agree or disagree with the following statements (1: Strongly disagree, 2: Somewhat disagree, 3: Neutral, 4: Somewhat agree, 5: Strongly agree):

1. Project team did a lot of redundant work (Coordination1).
2. Project team did a lot of rework (Coordination2).
3. Project team had substantial roadblocks in collaboration (Coordination3).
4. Project team could not figure out how to solve problems when they came up (Coordination4).
5. Project team was never certain whether the work we were doing would need further rework (Coordination5).
6. Many tasks were out of control (Coordination6).

### *V. Software Process Success*

Please indicate the extent to which you agree or disagree with the following statements (1: Strongly disagree, 2: Somewhat disagree, 3: Neutral, 4: Somewhat agree, 5: Strongly agree):

1. This project had time overrun according to the original schedule (ProcSuccess1).
2. This project had cost overrun according to the original budget (ProcSuccess2).
3. This project cost less than if it had been developed locally (ProcSuccess3).
4. This project took less time than if it had been developed locally (ProcSuccess4).
5. The project fully utilized globally distributed project resources (ProcSuccess5).
6. This project fully took advantage of globally distributed IT talent (ProcSuccess6).

TABLE 7
The Effects of Different Measures of Global Team Boundaries on Coordination Effectiveness

| Measures of Team Boundaries | Model 1b | | |
|---|---|---|---|
| | Unstandardized coefficient | Std. Error | Sig. |
| Dispersion over multiple boundaries | -0.950 | 0.289 | 0.002 |
| Dispersion over location boundary | -0.534 | 0.220 | 0.018 |
| Dispersion over time zone boundary | -0.323 | 0.207 | 0.124 |
| Dispersion over cultural boundary | -0.557 | 0.232 | 0.020 |
| Dispersion over org. boundary | -0.623 | 0.215 | 0.005 |
| Number of locations | 0.006 | 0.004 | 0.147 |
| Number of time zones | 0.048 | 0.029 | 0.105 |
| Number of cultures | 0.007 | 0.009 | 0.439 |
| Number of organizations | -0.013 | 0.024 | 0.590 |
| Number of all boundaries | 0.004 | 0.003 | 0.183 |

### B. Measurement Items for the Project Stakeholders Survey

*Software success*

Please indicate the extent to which you agree or disagree with the following statements (1: Strongly disagree, 2: Somewhat disagree, 3: Neutral, 4: Somewhat agree, 5: Strongly agree):

1. This project delivered a software system meeting user requirements (SoftwareSuccess1).
2. This project delivered a software system with many defects (SoftwareSuccess2).
3. This project delivered a software system meeting functionality goals (SoftwareSuccess3).
4. This project delivered a software system meeting technical requirements/specifications (SoftwareSuccess4).
5. This project delivered a software system providing useful information for users (SoftwareSuccess5).
6. This project delivered an easy-to-use software system (SoftwareSuccess6).
7. Users were satisfied with the software system delivered by this project (SoftwareSuccess7).
8. This project delivered a software system resulting in the expected organizational benefits (SoftwareSuccess8).

### C. Factor Loadings and Cross-Loadings for Reflective Constructs

See Table 6.

### D. The Effects of Different Measuares of Global Team Boundaries on Coordination Effectiveness

See Table 7.

## Acknowledgments

This work was supported in part by research grants from the Kogod School of Business at American University, Washington, DC, and its Center for IT and the Global Economy (CITGE).

## References

[1] J.A. Espinosa, S.A. Slaughter, R.E. Kraut, and J.D. Herbsleb, "Team Knowledge and Coordination in Geographically Distributed Software Development," *J. Management Information Systems,* vol. 24, pp. 135-169, 2007.
[2] F. Karamouzis, "Key Issues for IT Services Sourcing 2011," Gartner Research Report, 2011.
[3] W. Orlikowski, "Knowing in Practice: Enacting a Collective Capability in Distributed Organizing," *Organization Science,* vol. 13, pp. 249-273, May/June 2002.
[4] J.A. Espinosa, J.N. Cummings, J.M. Wilson, and B.M. Pearce, "Team Boundary Issues across Multiple Global Firms," *J. Management Information Systems,* vol. 19, pp. 157-190, Spring 2003.
[5] K. Fryer and M. Gothe, "Global Software Development and Delivery: Trends and Challenges," IBM, 2008.
[6] L. Argote, C.A. Insko, N. Yovetich, and A.A. Romero, "Group Learning Curves: The Effects of Turnover and Task Complexity on Group Performance," *J. Applied Social Psychology,* vol. 25, pp. 512-529, 1995.
[7] R. Banker, G. Davis, and S.A. Slaughter, "Software Development Practices, Software Complexity, and Software Maintenance Performance: A Field Study," *Management Science,* vol. 44, pp. 433-450, Apr. 1998.
[8] R.D. Banker and S.A. Slaughter, "The Moderating Effects of Structure on Volatility and Complexity in Software Enhancement," *Information Systems Research,* vol. 11, pp. 219-240, Sept. 2000.
[9] D.J. Campbell, "Task Complexity: A Review and Analysis," *Academy of Management Rev.,* vol. 13, pp. 40-52, Jan. 1988.
[10] D.P. Darcy, C.F. Kemerer, S.A. Slaughter, and J.E. Tomayko, "The Structural Complexity of Software: An Experimental Test," *IEEE Trans. Software Eng.,* vol. 31, no. 11, pp. 982-995, Nov. 2005.
[11] C. Kemerer, "Software Complexity and Software Maintenance: A Survey of Empirical Research," *Annals of Software Eng.,* vol. 1, pp. 1-22, 1995.
[12] R.E. Wood, "Task Complexity: Definition of the Construct," *Organizational Behavior and Human Decision Processes,* vol. 37, pp. 60-82, 1986.
[13] Y. Ren, S. Kiesler, and S.R. Fussell, "Multiple Group Coordination in Complex and Dynamic Task Environments: Interruptions, Coping Mechanisms, and Technology Recommendations," *J. Management Information Systems,* vol. 25, pp. 105-130, Summer 2008.
[14] W. Xia and G. Lee, "Grasping the Complexity of IS Development Projects," *Comm. ACM,* vol. 47, pp. 68-74, 2004.
[15] W. Xia and G. Lee, "Complexity of Information Systems Development Projects: Conceptualization and Measurement Development," *J. Management Information Systems,* vol. 22, pp. 45-83, 2005.
[16] J.A. Espinosa, J. Cummings, and C. Pickering, "Time Separation, Coordination, and Performance in Technical Teams," *IEEE Trans. Eng. Management,* vol. 59, no. 1, pp. 91-103, Feb. 2012.
[17] T. Malone and K. Crowston, "The Interdisciplinary Study of Coordination," *ACM Computing Surveys,* vol. 26, pp. 87-119, 1994.
[18] J.D. Herbsleb and A. Mockus, "An Empirical Study of Speed and Communication in Globally Distributed Software Development," *IEEE Trans. Software Eng.,* vol. 29, no. 6, pp. 481-494, June 2003.
[19] P.J. Hinds and D.E. Bailey, "Out of Sight, Out of Synch: Understanding Conflict in Distributed Teams," *Organization Science,* vol. 14, pp. 615-632, Nov./Dec. 2003.
[20] B. Latane, J.H. Liu, A. Nowak, M. Bonevento, and L. Zheng, "Distance Matters: Physical Space and Social Impact," *Personality and Social Psychology Bull.,* vol. 21, pp. 795-805, 1995.
[21] G.M. Olson and J.S. Olson, "Distance Matters," *Human-Computer Interaction,* vol. 15, pp. 139-179, 2000.
[22] K. Crowston and E.E. Kammerer, "Coordination and Collective Mind in Software Requirements Development," *IBM Systems J.,* vol. 37, pp. 227-245, 1998.
[23] C. Larman, *Agile & Iterative Development: A Manager's Guide.* Addison-Wesley, 2004.
[24] V. Sambamurthy and L.J. Kirsch, "An Integrative Framework of the Information Systems Development Process," *Decision Sciences,* vol. 31, pp. 391-411, 2000.
[25] B. Curtis, M. Kellner, and J. Over, "Process Modeling," *Comm. ACM,* vol. 35, pp. 75-90, 1992.
[26] G. DeSanctis and M.S. Poole, "Capturing the Complexity in Advanced Technology Use: Adaptive Structuration Theory," *Organization Science,* vol. 5, pp. 121-145, 1994.

[27] G. Lee, W.H. DeLone, and J.A. Espinosa, "Flexibility and Rigor: Ambidextrous Coping Strategies in Globally-Distributed Software Development Projects," *Comm. ACM,* vol. 49, pp. 35-40, 2006.

[28] J.D. Herbsleb and A. Mockus, "An Empirical Study of Speed and Communication in Globally-Distributed Software Development," *IEEE Trans. Software Eng.,* vol. 29, no. 6, pp. 481-494, June 2003.

[29] J.A. Espinosa, S.A. Slaughter, R.E. Kraut, and J.D. Herbsleb, "Familiarity, Complexity, and Team Performance in Geographically Distributed Software Development," *Organization Science,* vol. 18, pp. 613-630, 2007.

[30] T.W. Malone and K. Crowston, "The Interdisciplinary Study of Coordination," *ACM Computing Surveys,* vol. 26, pp. 87-119, 1994.

[31] R.D. Banker and S.A. Slaugher, "The Moderating Effects of Structure on Volatility and Complexity in Software Enhancement," *Information Systems Research,* vol. 11, pp. 219-240, Sept. 2000.

[32] J.D. Thompson, *Organizations in Action.* McGraw Hill, 1967.

[33] A.H. Van de Ven, L.A. Delbecq, and R.J. Koenig, "Determinants of Coordination Modes within Organizations," *Am. Sociological Rev.,* vol. 41, pp. 322-338, 1976.

[34] J.G. March and H.A. Simon, *Organizations.* John Wiley & Sons, 1958.

[35] R.E. Kraut and L.A. Streeter, "Coordination in Software Development," *Comm. ACM,* vol. 38, pp. 69-81, 1995.

[36] T.J. Allen, *Managing the Flow of Technology.* MIT Press, 1977.

[37] J. Cummings, "Work Groups, Structural Diversity, and Knowledge Sharing in a Global Organization," *Management Science,* vol. 50, pp. 352-3641, Mar. 2004.

[38] M.B. O'Leary and M. Mortensen, "Go (Con)figure: Subgroups, Imbalance, and Isolates in Geographically Dispersed Teams," *Organization Science,* vol. 21, pp. 115-131, 2009.

[39] J.A. Espinosa, G. Lee, and W.H. DeLone, "Global Boundaries, Task Processes and IS Project Success: A Field Study," *Information Technology and People,* vol. 19, pp. 345-370, 2006.

[40] J. Cummings, J.A. Espinosa, and C. Pickering, "Crossing Spatial and Temporal Boundaries in Globally Distributed Projects: A Relational Model of Coordination Delay," *Information Systems Research,* vol. 20, pp. 420-439, 2009.

[41] M. Lu, M.B. Watson-Manheim, K.M. Chudoba, and E. Wynn, "How Does Virtuality Affect Team Performance in a Global Organization? Understanding the Impact of Variety of Practices," *J. Global Information Technology Management,* vol. 9, pp. 4-23, 2006.

[42] S.D. Teasley, L.A. Covi, M.S. Krishnan, and J.S. Olson, "Rapid Software Development through Team Collocation," *IEEE Trans. Software Eng.,* vol. 28, no. 7, pp. 671-683, July 2002.

[43] D. Damian, F. Lanubile, and T. Mallardo, "On the Need for Mixed Media in Distributed Requirements Negotiations," *IEEE Trans. Software Eng.,* vol. 34, no. 1, pp. 116-132, Jan./Feb. 2008.

[44] M. Jarke, "Requirements Tracing," *Comm. ACM,* vol. 41, pp. 32-36, 1998.

[45] R.B. Duncan, "Characteristics of Organizational Environments and Perceived Environmental Uncertainty," *Administrative Science Quarterly,* vol. 17, pp. 313-327, 1972.

[46] B. Ramesh, L. Cao, K. Mohan, and P. Xu, "Can Distributed Software Development Be Agile?" *Comm. ACM,* vol. 49, pp. 41-46, 2006.

[47] D.E. Harter, C.F. Kemerer, and S.A. Slaughter, "Does Software Process Improvement Reduce the Severity of Defects? A Longitudinal Field Study," *IEEE Trans. Software Eng.,* vol. 38, no. 4, pp. 810-827, July/Aug. 2012.

[48] M. Unterkalmsteiner, T. Gorschek, A.K.M.M. Islam, C.K. Cheng, R.B. Permadi, and R. Feldt, "Evaluation and Measurement of Software Process Improvement—A Systematic Literature Review," *IEEE Trans. Software Eng.,* vol. 38, no. 2, pp. 398-424, Mar./Apr. 2012.

[49] B. Ramesh, K. Mohan, and L. Cao, "Ambidexterity in Agile Distributed Development: An Empirical Investigation," *Information Systems Research,* vol. 23, pp. 323-339, 2012.

[50] B.W. Boehm and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed.* Addison-Wesley, 2004.

[51] J. Birkinshaw and C.B. Gibson, "Building Ambidexterity into an Organization," *MIT Sloan Management Rev.,* vol. 45, pp. 47-55, 2004.

[52] P.J. Agerfalk, B. Fitzgerald, and S.A. Slaughter, "Flexible and Distributed Information Systems Development: State of the Art and Research Challenges," *Information Systems Research,* vol. 20, pp. 317-328, 2009.

[53] A.B. Pyster and R.H. Thayer, "Software Engineering Project Management 20 Years Later," *IEEE Software,* vol. 22, no. 5, pp. 18-21, Sept./Oct. 2005.

[54] P. Jalote, *CMM in Practice: Processes for Executing Software Projects at Infosys.* Addison-Wesley, 2000.

[55] D. Berry, C. Hungate, and T. Temple, "Delivering Expected Value to Users and Stakeholders with User Engineering," *IEEE Software,* vol. 20, no. 4, pp. 542-567, 2003.

[56] S.R. Nidumolu, "Standardization, Requirements Uncertainty and Software Project Performance," *Information & Management,* vol. 31, pp. 135-150, 1996.

[57] I. Oshri, P.v. Fenema, and J. Kotlarsky, "Knowledge Transfer in Globally Distributed Teams: The Role of Transactive Memory," *Information Systems J.,* vol. 18, pp. 593-616, 2008.

[58] C.D. Cramton, "The Mutual Knowledge Problem and Its Consequences for Dispersed Collaboration," *Organization Science,* vol. 12, pp. 346-371, 2001.

[59] R.J. Klimoski and S. Mohammed, "Team Mental Model: Construct or Metaphor," *J. Management,* vol. 20, pp. 403-437, 1994.

[60] W.S. Humphrey, T.R. Snyder, and R.R. Willis, "Software Process Improvement at Hughes Aircraft," *IEEE Software,* vol. 8, no. 4, pp. 11-23, July 1991.

[61] K. Lyytinen and G.M. Rose, "Information System Development Agility as Organizational Learning," *European J. Information Systems,* vol. 15, pp. 183-199, 2006.

[62] G. Lee and W. Xia, "The Ability of Information Systems Development Project Teams to Respond to Business and Technology Changes: A Study of Flexibility Measures," *European J. Information Systems,* vol. 14, pp. 75-92, 2005.

[63] G. Lee and W. Xia, "Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility," *MIS Quarterly,* vol. 34, pp. 87-114, 2010.

[64] K. Conboy, "Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development," *Information Systems Research,* vol. 20, pp. 329-354, 2009.

[65] M.A. Cusumano, "Managing Software Development in Globally Distributed Teams," *Comm. ACM,* vol. 51, pp. 15-17, 2008.

[66] S. Sarker and S. Sarker, "Exploring Agility in Distributed Information Systems Development Teams: An Interpretive Study in an Offshoring Context," *Information Systems Research,* vol. 20, pp. 440-461, 2009.

[67] P.J. Agerfalk and B. Fitzgerald, "Flexible and Distributed Software Processes: Old Petunias in New Bowls?" *Comm. ACM,* vol. 49, pp. 27-34, 2006.

[68] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change,* second ed. Addison-Wesley, 2005.

[69] A. Cockburn, *Agile Software Development: The Cooperative Game,* second ed. Addison-Wesley, 2007.

[70] M. van Oosterhout, E. Waarts, and J. van Hillegersberg, "Change Factors Requiring Agility and Implications for IT," *European J. Information Systems,* vol. 15, pp. 132-145, 2006.

[71] S. Slaughter, L. Levine, B. Ramesh, R. Baskerville, and J. Pries-Heje, "Aligning Software Processes and Strategy," *MIS Quarterly,* vol. 30, pp. 891-918, 2006.

[72] S.R. Nidumolu and G.W. Knotts, "The Effects of Customizability and Reusability on Perceived Process and Competitive Performance of Software Firms," *MIS Quarterly,* vol. 22, pp. 105-137, 1998.

[73] P. Adler, S., B. Goldoftas, and D.I. Levine, "Flexibility versus Efficiency? A Case Study of Model Changeovers in the Toyota Production System," *Organization Science,* vol. 10, pp. 43-68, 1999.

[74] S. Sarker and S. Sahay, "Implications of Space and Time for Distributed Work: An Interpretive Study of US-Norwegian Systems Development Teams," *European J. Information Systems,* vol. 13, pp. 3-20, 2004.

[75] B. Fitzgerald, N.L. Russo, and T. O'Kane, "Software Development Method Tailoring at Motorola," *Comm. ACM,* vol. 46, pp. 64-70, 2003.

[76] J. Cameron, "Configurable Development Processes," *Comm. ACM,* vol. 45, pp. 72-77, 2002.

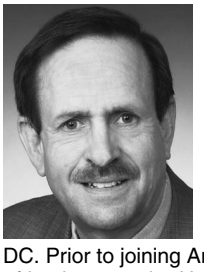
[77] P. Xu and B. Ramesh, "Software Process Tailoring: An Empirical Investigation," *J. Management Information Systems,* vol. 24, pp. 293-328, 2007.

[78] S.R. Nidumolu, "A Comparison of the Structural Contingency and Risk-Based Perspectives on Coordination in Software-Development Projects," *J. Management Information Systems,* vol. 13, pp. 77-113, 1996.

[79] S.R. Nidumolu, "The Effect of Coordination and Uncertainty on Software Project Performance: Residual Performance Risk as an Intervening Variable," *Information Systems Research,* vol. 6, pp. 191-219, 1995.
[80] M.S. Krishnan and M.I. Kellner, "Measuring Process Consistency: Implications for Reducing Software Defects," *IEEE Trans. Software Eng.,* vol. 25, no. 6, pp. 800-815, Nov./Dec. 1999.
[81] D.M. Ahern, A. Clouse, and R. Turner, *CMMI Distilled: A Practical Introduction to Integrated Process Improvement,* third ed. Addison-Wesley Professional, 2008.
[82] M. Cataldo and J.D. Herbsleb, "Coordination Breakdowns and Their Impact on Development Productivity and Software Failures," *IEEE Trans. Software Eng.,* vol. 39, no. 3, pp. 343-360, Mar. 2013.
[83] C. Deephouse, T. Mukhopadhyay, D.R. Goldenson, and M.I. Keller, "Software Processes and Project Performance," *J. Management Information Systems,* vol. 12, pp. 187-205, 1996.
[84] W.H. DeLone and E.R. McLean, "The DeLone and McLean Model of Information Systems Success: A Ten-Year Update," *J. Management Information Systems,* vol. 19, pp. 9-30, 2003.
[85] R. Dorfman, "A Formula for the Gini Coefficient," *Rev. Economics and Statistics,* vol. 61, pp. 146-156, 1979.
[86] D. Miller and P.H. Friesen, "Innovation in Conservative and Entrepreneurial Firms: Two Models of Strategic Momentum," *Strategic Management J.,* vol. 3, pp. 1-25, 1982.
[87] J.L. Whitten, L.D. Bentley, and K.C. Dittman, *System Analysis and Design Methods.* McGraw-Hill, 2001.
[88] J.G. Cooprider and J.C. Henderson, "Technology-Process Fit: Perspectives on Achieving Prototyping Effectiveness," *J. Management Information Systems,* vol. 7, pp. 67-87, 1991.
[89] W.H. DeLone and E.R. McLean, "Information Systems Success: The Quest for the Dependent Variable," *Information Systems Research,* vol. 3, pp. 60-95, 1992.
[90] S. Petter, D. Straub, and A. Rai, "Specifying Formative Constructs in Information Systems Research," *MIS Quarterly,* vol. 31, pp. 623-656, 2007.
[91] K.D. Loch, D.W. Straub, and S. Kamel, "Diffusing the Internet in the Arab World: The Role of Social Norms and Technological Culturation," *IEEE Trans. Eng. Management,* vol. 50, no. 1, pp. 45-63, Feb. 2003.
[92] N.K. Malhotra, S.S. Kim, and A. Patil, "Common Method Variance in IS Research: A Comparison of Alternative Approaches and a Reanalysis of Past Research," *Management Science,* vol. 52, pp. 1865-1883, 2006.
[93] P.M. Podsakoff and D.W. Organ, "Self-Reports in Organizational Research: Problems and Prospects," *J. Management,* vol. 12, pp. 531-544, 1986.
[94] M.A. Korsgaard and L. Roberson, "Procedural Justice in Performance Evaluation: The Role of Instrumental and Non-instrumental voice in Performance Appraisal Discussions," *J. Management,* vol. 21, pp. 657-669, 1995.
[95] L. Aiken and S. West, *Multiple Regression: Testing and Interpreting Interactions.* Sage, 1992.
[96] D.A. Belsley, E. Kuh, and R.E. Welsch, *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity.* John Wiley & Sons, 1980.
[97] L.M. Maruping, V. Venkatesh, and R. Agarwal, "A Control Theory Perspective on Agile Methodology Use and Changing User Requirements," *Information Systems Research,* vol. 20, pp. 377-399, 2009.
[98] M.L. Harris, R.W. Collins, and A.R. Hevner, "Control of Flexible Software Development under Uncertainty," *Information Systems Research,* vol. 20, pp. 400-419, 2009.
[99] O.N. Huy, "Emotional Balancing of Organizational Continuity and Radical Change: The Contribution of Middle Managers," *Administrative Science Quarterly,* vol. 47, pp. 31-69, 2002.
[100] D. Ancona and D. Caldwell, "Demography and Design: Predictors of New Product Team Performance," *Organization Science,* vol. 3, pp. 321-341, 1992.
[101] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences,* second ed. Routledge Academic, 1988.

**Gwanhoo Lee** received the BS and MS degrees in industrial engineering from Seoul National University and the PhD degree in information systems from the University of Minnesota. He is currently an associate professor of information technology and the director of Center for IT and the Global Economy in the Kogod School of Business at the American University, Washington, DC. His current research interests include complexity and agility issues in information systems development and software engineering, global teams, and government use of ICT. He has published his research in premier journals and conference proceedings. He has received grants from the IBM Center for the Business of Government and the Juran Center for Leadership in Quality. He is an associate editor of the *Asia Pacific Journal of Information Systems* and a co-president of Korean Association of Information Systems. He is a member of the Association for Information Systems, the Society for Information Management, and the Academy of Management.

**J. Alberto Espinosa** received the BS degree in mechanical engineering from Pontificia Universidad Catolica, Peru, the MBA degree from Texas Tech University, and the MS and PhD degrees in information systems from the Tepper School of Business at Carnegie Mellon University. He is the chair of the Information Technology Department and a full professor at the Kogod School of Business, American University. His research focuses on coordination and performance in global technical projects across global boundaries, particularly spatial and temporal distance. His current research areas include coordination of knowledge work across time zones and coordination in large-scale technical collaboration tasks like enterprise architecture. His research has been published in several leading scholarly journals and conference proceedings and he is an associate editor of Information Systems Research.



**William H. DeLone** received the BS degree in mathematics from Villanova University, the MS degree in industrial administration from Carnegie-Mellon University, and the PhD degree in computer information systems from the University of California Los Angeles. He is currently a professor of information technology and the executive director of the Center for IT and the Global Economy in the Kogod School of Business at the American University, Washington, DC. Prior to joining American University, he was an associate professor of business at the University of the Virgin Islands. His current research interests include the assessment of information systems' effectiveness and value, e-government and public value, the management of global software development, and the effective deployment of information and communications technology in developing countries. He has published his research in premier journals and conference proceedings. He is a member of the Association for Information Systems, the Society for Information Management, and INFORMS.