MIS Quarterly

RESEARCH ARTICLE

TOWARD AGILE: AN INTEGRATED ANALYSIS OF QUANTITATIVE AND QUALITATIVE FIELD DATA ON SOFTWARE DEVELOPMENT AGILITY[1]

By: **Gwanhoo Lee**
Department of Information Technology
Kogod School of Business
American University
4400 Massachusetts Avenue NW
Washington, DC 20016-8044
U.S.A.
glee@american.edu

Weidong Xia
Department of Decision Sciences and Information Systems
College of Business Administration
Florida International University
11200 SW 8 Street
Miami, FL 33199
Weidong.Xia@fiu.edu

Abstract

As business and technology environments change at an unprecedented rate, software development agility to respond to changing user requirements has become increasingly critical for software development performance. Agile software development approaches, which emphasize sense-and-respond, self-organization, cross-functional teams, and continuous adaptation, have been adopted by an increasing number of organizations to improve their software development agility. However, the agile development literature is largely anecdotal and prescriptive, lacking empirical evidence and theoretical foundation to support the principles and practices of agile development. Little research has empirically examined the software development agility construct in terms of its dimensions, determinants, and effects on software development performance. As a result, there is a lack of understanding about how organizations can effectively implement an agile development approach.

Using an integrated research approach that combines quantitative and qualitative data analyses, this research opens the black box of agile development by empirically examining the relationships among two dimensions of software development agility (software team response extensiveness and software team response efficiency), two antecedents that can be controlled (team autonomy and team diversity), and three aspects of software development performance (on-time completion, on-budget completion, and software functionality). Our PLS results of survey responses of 399 software project managers suggest that the relationships among these variables are more complex than what has been perceived by the literature. The results suggest a tradeoff relationship between response extensiveness and response efficiency. These two agility dimensions impact software development performance differently: response efficiency positively affects all of on-time completion, on-budget completion, and software functionality, whereas response extensiveness positively affects only software functionality. The results also suggest that team autonomy has a positive effect on response efficiency and a negative effect on response extensiveness, and that team diversity has a positive effect on response extensiveness. We conducted 10 post hoc case studies to qualitatively cross-validate our PLS results and provide rich, additional insights regarding the complex, dynamic interplays between auton-

[1]Detmar Straub was the accepting senior editor for this paper. Bill Kettinger served as the associate editor.

omy, diversity, agility, and performance. The qualitative analysis also provides explanations for both supported and unsupported hypotheses. We discuss these qualitative analysis results and conclude with the theoretical and practical implications of our research findings for agile development approaches.

Keywords: Software development agility, agile software development, team autonomy, team diversity, software development performance, requirement change, partial least square, case study

Introduction

The unprecedented rate of change in business and technology has made it increasingly difficult for software teams to determine user requirements and respond to their changes (Schmidt et al. 2001). A U.S. Department of Defense study shows that 45 percent of software features fail to meet user needs and requirements (Larman 2004). Agile software development approaches such as XP (eXtreme Programming), Scrum, DSDM (Dynamic Systems Development Method), and FDD (Feature-Driven Development) have been proposed as solutions to improve a software team's ability to embrace and respond to changing requirements (Beck and Andres 2005; Coad et al. 1999; Schwaber and Beedle 2002; Stapleton 1997). Agile development approaches differ from the traditional, plan-driven, structured approaches as the former put more emphasis on lean processes and dynamic adaptation than on detailed front-end plans and heavy documentation (Nerur and Balijepally 2007).

At the heart of agile development approaches is the notion of software development agility, which is defined in this research as a software team's ability to efficiently and effectively respond to user requirement changes. However, agility is difficult to achieve in practice (Cockburn 2001). It has been reported that only 11 percent of IS organizations were able to keep up with business demands and that 76 percent were not able to effectively cope with changing business needs (Koch 2006). This lack of agility often results in substantial financial loss (Austin and Devin 2003).

Despite the growing popularity and importance of agile approaches, little research has empirically examined their key concepts and underlying theoretical relationships (Baskerville 2006; Boehm and Turner 2004; Larman 2004). The core values, principles, and practices of agile development have been derived mainly from past experiences and its effectiveness has been supported largely by anecdotal evidence and rhetorical arguments. Furthermore, the concept of software development agility has not been well understood. As a result, many organizations adopt agile development approaches without clearly understanding how agility is defined and measured and what factors they can control to influence it. Highsmith (2000) argues that "techniques without a theoretical base are reduced to a series of steps executed by rote" (p. 14). Unfortunately, if, how, why, and when agile development works or doesn't work remains largely a black box. This research aims to fill this literature gap by addressing critical questions pertaining to agile development approaches. As software development agility is a central concept and a core value of agile development (Agile Alliance 2001; Larman 2004), we investigate its dimensions, determinants, and effects.

We intend to make the following contributions. First, while prior agile development literature has not explicitly distinguished different dimensions of software development agility, we propose that it is not a monolithic, single dimensional construct; rather, it is a multidimensional construct comprised of different and even conflicting capabilities. We identify and assess two key agility dimensions, namely software team response extensiveness and software team response efficiency, which tap into different important aspects of agility. Indeed, agile development approaches invariably promote extensive responses to requirement changes in a rapid and efficient manner (Erickson et al. 2005; Henderson-Sellers and Serour 2005; Lyytinen and Rose 2006). Software team response extensiveness is defined as the proportion of various types of changing user requirements that a software team responds to and incorporates into the software system. For example, if a software team incorporates 80 out of 100 different requirement changes, the team's response extensiveness would be 80 percent. Greater response extensiveness indicates greater software development agility. On the other hand, software team response efficiency is defined as the minimal time, cost, personnel, and resources that the team requires to respond to and incorporate a particular requirement change. Software development is considered agile when the team requires relatively little time, cost, personnel, and resources to respond to a requirement change. An important question that this research addresses is: How are these two dimensions of software development agility related to each other?

Second, we examine how team autonomy and team diversity, two team-level variables organizations can control in staffing and managing projects, affect software development agility. In this research, team autonomy refers to the extent to which the software team is empowered with the authority and control in making decisions to carry out the project. Team diversity refers to the extent to which team members are different

in terms of their functional backgrounds, skills, expertise, and work experience. Agile development views team autonomy and team diversity as important principles and practices that foster software development agility (Larman 2004). For example, agile development advocates self-organizing, self-directed, and self-disciplined teams (Highsmith 2004; Larman 2004). Furthermore, agile development posits that a software team should have a variety of skills and perspectives, necessary for sensing problems and pitfalls, thinking of multiple ways to solve problems, and implementing solutions (Beck and Andres 2005). To our knowledge, no research has empirically validated how team autonomy and team diversity influence software development agility.

Third, this research examines how the two dimensions of software development agility affect software development performance in terms of on-time completion, on-budget completion, and software functionality. The positive effect of software development agility on performance has been supported in the literature mainly by anecdotal evidence and rhetorical arguments. Thus, rigorous empirical investigation is required to answer the question of if and how software development agility affects development performance.

To address these emergent, complex issues associated with software development agility, we use an integrated multi-method approach that combines quantitative and qualitative methods. This integrated approach intends to formulate research questions and develop measurement instruments grounded in field data, provide both statistical evidence and rich explanations, triangulate results, and discover novel insights that stimulate further research (Kaplan and Duchon 1988; Lee 1991; Mingers 2001). Our research starts with preliminary qualitative field studies and focus groups that help formulate research problems and questions, identify key constructs, and develop measures for new constructs. We then conduct a PLS analysis of the data from a large-scale quantitative survey to validate measurement and test hypotheses. Finally, we conduct multiple mini-case studies to cross-validate the PLS results, provide rich, additional insights, and offer explanations for both supported and unsupported hypotheses.

Theoretical Background

Agile Software Development Approaches

Agile software development approaches have evolved since the mid-1990s as new alternative solutions to the inability of traditional "heavyweight" methods to address such enduring problems as time/cost overruns and the lack of responsiveness to changing requirements (Beck and Andres 2005; Boehm and Turner 2004; Cockburn 2001; Highsmith 2004; Larman 2004). Agile development views the software development process to be dynamic, evolving, and organic, rather than static, predefined, and mechanistic (Beck and Andres 2005; Highsmith 2000). Commonly used agile development methods include XP (eXtreme Programming), Scrum, DSDM (Dynamic Systems Development Method), and FDD (Feature-Drive Development). In 2001, the 4 core values and 12 principles of agile development were formally introduced and endorsed in the publication of the *Agile Manifesto* by some of the prominent members of the agile development community. Since then, agile development has attracted much interest from the software industry (Dybå and Dingsøyr 2008).

According to the Agile Manifesto, agile development values individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan (Agile Alliance 2001). It employs "light and barely sufficient" methods to minimize time-consuming and costly software processes such as detailed front-end planning and heavy documentation (Boehm and Turner 2004; Fitzgerald et al. 2006). Agile development attempts to effectively manage volatile and changing user requirements through a variety of practices and techniques (Beck and Andres 2005). It promotes frequent and continuous delivery of working software, embracing changing requirements, close collaboration between developers and users, self-organizing and empowered teams, face-to-face communication, technical excellence, simplicity, and continuous adaptation (Agile Alliance 2001). Embracing and responding to changing user requirements is at the heart of agile development.

Although many benefits of agile development have been speculated and claimed, and increasingly more organizations are adopting the approach, there have been few empirical field studies that have rigorously examined if, how, and why agile development is effective (Fruhling and De Vreede 2006; Moe et al. 2008). As a result, agile development lacks theoretical underpinnings and scientific evidence that support its claimed benefits and key principles (Erickson et al. 2005). To fill this gap, researchers have called for structured, rigorous empirical studies on agile development with a common research agenda (Dybå and Dingsøyr 2008).

Software Development Agility, Autonomy, and Diversity

We review the prior agile development literature relevant to our main research constructs: software development agility,

team autonomy, and team diversity. Table 1 summarizes the definitions and descriptions pertaining to the constructs and Table 2 summarizes the key principles and practices underlying the Agile Manifesto and four commonly used agile methods that emphasize software development agility, team autonomy, and team diversity (Beck and Andres 2005; Cockburn 2001; Highsmith 2004; Larman 2004).

As shown in Table 1, prior literature provides various definitions and descriptions of software development agility. While the literature is vague about the underlying dimensions and measures of software development agility, there is a common theme underlying the various definitions and descriptions in that agility is generally defined in terms of embracing and responding to change (Conboy and Fitzgerald 2004; Erickson et al. 2005; Henderson-Sellers and Serour 2005; Highsmith 2004; Larman 2004; Qumer and Henderson-Sellers 2008). Similarly, we define software development agility in this research as the software team's capability to efficiently and effectively respond to and incorporate user requirement changes during the project life cycle.

Furthermore, it appears that prior literature tends to view software development agility as consisting of two important elements that correspond to our conceptualization of the two agility dimensions: response extensiveness and response efficiency. Response extensiveness relates to the extent, range, scope, or variety of software team responses. In contrast, response efficiency relates to the time, cost, resources, or effort associated with software team responses. As summarized in Table 1, agile development promotes both response extensiveness in terms of embracing various changes (Henderson-Sellers and Serour 2005; Qumer and Henderson-Sellers 2008) and response efficiency in terms of doing so with high speed and low cost (Conboy and Fitzgerald 2004; Erickson et al. 2005; Larman 2004; Qumer and Henderson-Sellers 2008). As shown in Table 2, software development agility is at the heart of agile development principles and practices. Agile development approaches promote agility through short, incremental, iterative, time-boxed development cycles, self-organizing teams, active participation of stakeholders, and continuous delivery of working software.

Team autonomy and team diversity have been invariably viewed by prior literature as important principles for improving software development agility, as shown in Tables 1 and 2. Agile development is fundamentally people-centric and recognizes the value of team members' competencies in bringing agility to development processes (Nerur and Balijepally 2007). It has been argued that getting the right people with appropriate skills and empowering them in decision-making are critical for agile development success (Chow and Cao 2008; Cockburn 2007; Highsmith 2004).

Team autonomy refers to the degree of discretion and independence granted to the team in scheduling the work, determining the procedures and methods to be used, selecting and deploying resources, hiring and firing team members, assigning tasks to team members, and carrying out assigned tasks (Breaugh 1985). It decentralizes decision-making power to those who will actually carry out the work (Tatikonda and Rosenthal 2000). As shown in Tables 1 and 2, agile development emphasizes the importance of autonomous, self-organizing, self-directed, self-disciplined software teams for achieving software development agility (Highsmith 2004; Nerur and Balijepally 2007; Sharp and Robinson 2004). Autonomous teams have considerable leeway in how they deliver results (Highsmith 2004). Autonomy brings decision-making authority to the hands of the people who face and handle problems every day, thus, it increases the speed and effectiveness of problem solving (Larman 2004; Tata and Prasad 2004). While team autonomy is one of the key principles in agile development, little empirical research has tested its effect on software development agility.

Team diversity is defined as the heterogeneity within the team in terms of individual attributes, such as age, gender, ethnic background, education, functional background, tenure, and technical abilities (Williams and O'Reilly 1998). As shown in Tables 1 and 2, agile development proposes that diverse software teams are more effective than homogeneous teams in sensing and responding to various environmental changes (Cockburn 2007; MacCormack et al. 2001). Drawing upon Ashby's law of requisite variety (Ashby 1956), the agile literature suggests that a software team's internal variety should match the variety and complexity of the environment and that the diversity of skills amplify the internal variety that enables the team to respond to the changing environment (Highsmith 2004; Nerur and Balijepally 2007). Although conflict is the inevitable companion of diversity, agile development suggests that software teams need to bring together a variety of skills and perspectives to see problems and pitfalls, to think of multiple ways to solve problems, and to implement the solutions (Coad et al. 1999). However, little empirical research has tested if and how team diversity affects software development agility.

Research Model and Hypotheses

Our research model is shown in Figure 1. The central constructs of the research model are software team response extensiveness and software team response efficiency that tap into two different, important dimensions of software development agility. Hypotheses 1 through 4 posit that software team autonomy and software team diversity have differential ef-

Table 1. Agile Development Literature on Agility, Autonomy and Diversity

Construct	Literature	Relevant Definitions/Concepts/Ideas
Software development agility	Conboy & Fitzgerald (2004)	Agility is defined as the continual readiness of an entity to rapidly or inherently, proactively or reactively, embrace change, through high-quality, simplistic, economical components and relationships with its environment
	Highsmith (2004)	Agility is the ability to both create and respond to change in order to profit in a turbulent business environment; it is the ability to balance flexibility and stability
	Larman (2004)	Agility is rapid and flexible response to change
	Erickson et al. (2005)	Agility is associated with such related concepts as nimbleness, suppleness, quickness, dexterity, liveliness, or alertness; it means to strip away the heaviness in traditional software development methodologies to promote quick response to changing environments and changes in user requirements
	Henderson-Seller & Serour (2005)	Agility refers to readiness for action or change; it has two dimensions: (1) the ability to adapt to various changes and (2) the ability to fine-tune and reengineer software development processes when needed
	Lyytinen & Rose (2006)	Agility is defined as the ability to sense and respond swiftly to technical changes and new business opportunities; it is enacted by exploration-based learning and exploitation-based learning
	Cockburn (2007)	Agility is being light, barely sufficient, and maneuverable
	Qumer & Henderson-Sellers (2008)	Agility is a persistent behavior or ability of an entity that exhibits flexibility to accommodate expected or unexpected changes rapidly, follows the shortest time span, and uses economical, simple, and quality instruments in a dynamic environment; agility can be evaluated by flexibility, speed, leanness, learning, and responsiveness
Team autonomy	Cockburn & Highsmith (2001)	Agile teams are characterized by self-organization
	Highsmith (2002)	Software teams should enable team decision-making
	Highsmith (2004)	The agile development supports self-organization, self-discipline, and self-management
	Larman (2004)	In Scrum, the team is empowered with the authority and resources to find their own way and solve their own problems
	Sharp & Robinson (2004)	Self-managing, self-organizing teams are essential for agile development culture, especially for XP
	Beck & Andres (2005)	One of the XP principles is team responsibility and authority
	Nerur & Balijepally (2007)	Self-organizing teams are key for responsiveness and flexibility
	Chow & Cao (2008)	Self-organizing teamwork is found to increase system quality
	Kelley (2008)	Empowerment is key for agile development
Team diversity	MacCormack et al. (2001)	Teams with greater amounts of broad experience are positively associated with project performance
	Highsmith (2004)	Getting the right people with appropriate skills is critical
	Beck & Andres (2005)	One of the XP principles is team diversity, which is enacted by the notion of "whole team"
	Cockburn (2007)	Team diversity is desirable; heterogeneous teams outperform homogeneous teams
	Nerur & Balijepally (2007)	Team diversity is key for agile development

Table 2. Key Principles and Practices of Agile Approaches/Methods

Agile Approach/ Method	Principles/Practices Emphasizing Software Development Agility	Principles/Practices Emphasizing Team Autonomy and Diversity
Agile Alliance Manifesto (Agile Alliance 2001)	• Welcome changing requirements, even late in development • Agile processes promote sustainable development • Deliver working software frequently • Continuous attention to technical excellence enhances agility	• The best architectures, requirements, and designs emerge from self-organizing teams • Build projects around motivated individuals; give them the environment and support they need, and trust them to get the job done • Teams reflect on how to become more effective and adjust their behavior • Business people and developers must work together daily
Scrum (Schwaber and Beedle 2002)	• Software team determines features of each sprint from an evolving product backlog • Create an increment of potentially shippable software during each sprint	• Teams determine how much of the features in the product backlog they want to commit to during the next sprint • Self-organizing, cross-functional teams across different phases/sprints
XP (Beck and Andres 2005)	• The highest priority is continuously satisfy changing customer needs • Rapid user review and feedback	• Align team authority/control with responsibility to get things done • Pair programming: two developers complement each other's skills and work
DSDM (Stapleton 1997)	• Development is iterative, incremental, and driven by user feedback • Delivering a perfect system is less important than delivering a system that addresses the current business needs	• Teams must be empowered to make project decisions without waiting for higher-level approval • Continuous interactions and cooperation among all project stakeholders
FDD (Coad et al. 1999)	• Customer/feature-centered iterative cycles • Regular build and inspection to ensure up-to-date systems	• Small, dynamically formed, autonomous teams are effective • Multiple cross-functional minds are always applied to each design decision

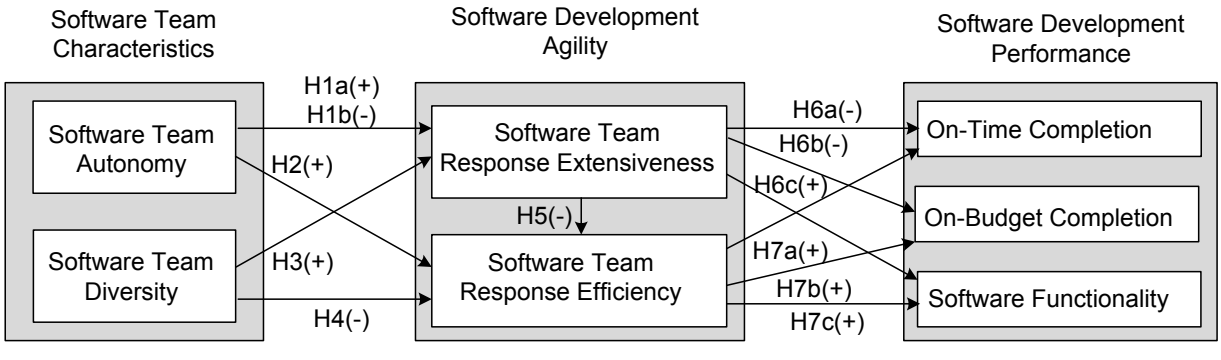

Figure 1. The Research Model

fects on software team response extensiveness and response efficiency. Hypothesis 5 posits a tradeoff relationship between software team response extensiveness and response efficiency. Finally, hypotheses 6 and 7 posit that software team response extensiveness and response efficiency have differential effects on three dimensions of software development performance: on-time completion, on-budget completion, and software functionality.

Effects of Team Autonomy on Software Development Agility

We propose two competing hypotheses for the effect of team autonomy on software team response extensiveness. On the one hand, autonomy facilitates creativity in solving problems and enhances team learning in uncertain environments (Imai et al. 1985). Self-organization and local control allow teams to be open to innovative ideas (Lyytinen and Rose 2006). Decentralized decision making enables autonomous teams to be effective in sensing and responding to environmental changes (McGrath 2001). To be adaptive and agile, a software team must be willing and able to take risks, and experiment through trial-and-error iterations. A higher degree of team autonomy is likely to lead to greater risk taking and experimentation (Tushman and O'Reilly 1996). An autonomous team is likely to freely experiment and search for solutions to a broad range of user requirement changes. Therefore, we propose

Hypothesis 1a. Software team autonomy positively affects the extensiveness of the team's response to user requirement change.

On the other hand, team autonomy may reduce the extensiveness of software team response. Software development typically requires software teams to make tradeoff decisions among the interdependent and conflicting goals of time, cost, and scope. If a team has a high level of autonomy, it has more latitude to say "no" to users' change requests, in order to meet time and cost goals. As a result, the team can be more selective in responding to changing requirements. In contrast, if a team has little autonomy, it may end up being an order taker and responding to every change request. In our preliminary field interviews, we found that highly autonomous teams were often selective in dealing with requirement change requests. Therefore, we propose the following competing hypothesis:

Hypothesis 1b. Software team autonomy negatively affects the extensiveness of the team's response to user requirement change.

Increased autonomy enables a software team to make and execute decisions at a higher speed with lower cost, because the team does not need to go through the bureaucratic organizational hierarchy, which is time-consuming and costly (Clark and Fujimoto 1991). An empowered, self-organized, autonomous team can sense and respond to requirement changes efficiently through direct and close interactions with users without waiting for managerial approval. As a result, increased autonomy allows the software team to reduce the time, cost, and resources required to sense requirement change needs and to make necessary changes to the system. Therefore, we propose

Hypothesis 2. Software team autonomy positively affects the efficiency of the team's response to user requirement change.

Effects of Team Diversity on Software Development Agility

Diversity can be a double-edged sword, improving group performance in certain tasks but, often, disrupting group processes (Milliken and Martins 1996; Pelled et al. 1999). Responding to a changing requirement is essentially a problem-solving process because a requirement change reflects a complex business and/or technical problem. To be agile, a software team should be able to develop effective solutions to various complex problems. According to the cognitive resource perspective, qualities such as a variety of expertise, experiences, backgrounds, and perspectives brought by diverse members increase the team's cognitive resources and ability to engage in complex problems (Aladwani 2002; Watson et al. 1993). Team members with diverse competencies and perspectives stimulate learning and innovation, and generate more alternative solutions for complex problems (Campion et al. 1993; Watson et al. 1993).

Furthermore, team members with diverse expertise and experiences can access diverse social networks and professional communities in their domains of expertise (Ancona and Caldwell 1992). The access to large external networks and communities can facilitate acquisition and development of new knowledge and skills that are necessary to respond to requirement changes. Diverse functional backgrounds help the software team understand the contexts of various change needs. Therefore, we expect a diverse team to be more capable of handling a wide range of requirement changes.

Hypothesis 3. Software team diversity positively affects the extensiveness of the team's response to user requirement change.

However, team diversity may negatively affect the efficiency of team response process. Social identity theory and self-categorization theory (Turner et al. 1987) suggest that, due to intergroup categorizations and different identities among workgroup members, diversity decreases team cohesion and integration (Webber and Donahue 2001), causes communication failures (Miller et al. 1998), and increases task-related conflict (Pelled et al. 1999). Diversity makes it difficult for team members to develop a shared mental model due to knowledge gaps within the team (Klimoski and Mohammed 1994; Mathieu et al. 2000). As a result, a diverse software team is likely to incur more time, cost, and efforts in communicating and coordinating tasks and making decisions to sense and understand change requests, to develop response strategies, and to implement appropriate responses. Therefore, we propose

Hypothesis 4. Software team diversity negatively affects the efficiency of the team's response to user requirement change.

Relationship Between Response Extensiveness and Response Efficiency

We propose that team response extensiveness negatively affects team response efficiency. Considering the impacts of requirement changes on time/cost/scope goals that are inherently conflicting, software teams tend to first choose how much they would respond to changes. This choice in turn affects response efficiency. One of the common practices underlying many agile development approaches is time boxing by which the software team balances its conflicting needs for embracing as many user requirement changes as possible and meeting time and cost goals at the same time (Larman 2004). For example, with Scrum, software teams first decide on the scope of the requirement changes in the user backlog that they need to address in the next sprint, and then implement them as efficiently as possible to meet project goals (Schwaber and Beedle 2002). Similarly, XP's weekly and quarterly development cycles enforce the same pattern; the customers first choose the scope of work for the next development cycle and then the software team implements it as efficiently as possible to meet the project goals (Beck and Andres 2005). Therefore, the agile approaches suggest that the choice of response extensiveness tends to precede response efficiency.

Furthermore, we argue that the more extensively the team responds to changes, the less efficient the team is in implementing each change. When a software team attempts to address a wide variety of requirement changes, the team is likely to respond not only to familiar or anticipated changes but also to unfamiliar or unanticipated changes. As a result, the team often needs to develop or acquire new knowledge and capabilities through search, experimentation, innovation, and variation. This process consumes a substantial amount of time, cost, and resources and reduces the team's attention to speed or cost, thus decreasing response efficiency (Lyytinen and Rose 2006). In one of our preliminary field interviews, we found that a software team tried to incorporate all user change requests, some of which related to issues with which the team was not familiar. As a result, the team was overwhelmed with the variety and amount of change requests and could not address any of the changes efficiently.

In contrast, narrow, selective, and controlled team responses to requirement changes would lead to higher response efficiency. When a software team selectively responds to certain types of user requirement changes, the team can refine, optimize, and streamline its response process through repeated implementations. As a result, the team is likely to reduce coordination and implementation costs for handling requirement changes, thus increasing response efficiency. Therefore, we propose

Hypothesis 5. Software team response extensiveness negatively affects software team response efficiency.

Effects of Software Development Agility on Software Project Performance

Both the traditional software development literature and the agile literature suggest that on-time completion, on-budget completion, and software functionality are important dimensions of software development performance (Highsmith 2004; Kerzner 2005; Mitchell 2006; Nidumolu 1995). On-time completion and on-budget completion refer to the extent to which a software project meets its baseline goals for duration and cost. Software functionality refers to the extent to which the delivered software system meets its functional goals, user needs, and technical requirements (Weitzel and Graen 1989). There are inherent tradeoffs among time, cost, and functionality because pursuing one often comes at the expense of the others (Nidumolu 1995).

We posit that extensive responses cause time and cost overruns. To extensively respond to many different requirement changes, software teams may need to acquire new resources and capabilities or reconfigure existing resources, processes, and capabilities, requiring substantial organizational learning and knowledge transfer. This requires a substantial amount of additional time, cost, and resources. On the other hand, we

posit that extensive software team responses positively affect software functionality. For example, a software team's responsiveness to user requests has been found to improve the correctness of system configuration (Gefen and Ridings 2002). Organizations often experience important business changes during software development that, in turn, require changes in user requirements. The functionality of the software system would not satisfy up-to-date user needs if the team fails to embrace important changes. Therefore, we propose

Hypothesis 6a. Software team response extensiveness negatively influences on-time completion of software development.

Hypothesis 6b. Software team response extensiveness negatively influences on-budget completion of software development.

Hypothesis 6c. Software team response extensiveness positively influences software functionality.[2]

Managing requirement changes accounts for an increasing portion of software development duration and cost in today's turbulent business environment. If the efficiency of team response is high, the amount of additional time and costs necessary for handling requirement changes is minimal. That, in turn, would help reduce time and cost overruns and meet the initial time and budget goals. Furthermore, we propose that efficient team responses improve the functionality of the delivered software system. As the software team repeatedly implements responses to similar types of requirement changes, not only does the team increase efficiency of their response process but also streamlines, optimizes, and perfects their work. In contrast, inefficient team responses are more likely to cause problems and errors in the work process. All else being equal, efficient team responses are expected to result in high quality software functionality that effectively satisfies user requirements. Therefore, we propose

Hypothesis 7a. Software team response efficiency positively influences on-time completion of software development.[3]

Hypothesis 7b. Software team response efficiency positively influences on-budget completion of software development.[4]

Hypothesis 7c. Software team response efficiency positively influences software functionality.

Research Methods

Research Process and Study Sample

The detailed description of our research process is provided in Appendix A. In summary, we used an integrated multimethod approach that includes both quantitative and qualitative data analyses, consisting of five phases: (1) preliminary field interviews, (2) survey data collection, (3) measurement validation, (4) hypothesis testing, and (5) post hoc case studies. Quantitative data were collected and analyzed in Phases 2, 3 and 4, whereas qualitative data were collected and analyzed in Phases 1 and 5. The integration of quantitative and qualitative approaches helps address limitations of each approach by providing both statistical objectivity and a deeper understanding of contexts (Kaplan and Duchon 1988; Lee 1991; Trauth and Jessup 2000).

Preliminary field interviews, focus groups, sorting procedure, and pilot tests were used in Phase 1 to formulate research problems and questions, identify key constructs, and generate and refine measurement items. The survey respondents in Phase 2 were members of the Information Systems Specific Interest Group of the Project Management Institute (PMI-ISSIG). The target respondents were 1,740 North American members who were project managers and had recently managed a software development project. After measures were validated in Phase 3, the PLS analysis of 399 survey responses in Phase 4 provided statistical evidence for the hypothesized relationships among the constructs. In Phase 5, post hoc case studies were conducted to validate the PLS results, provide richer explanations and insights for the results, and reveal the complex dynamics of tensions among the constructs. We selected the cases that demonstrated diverse project profiles in terms of system type, project size, development methods, and project performance as well as different patterns of team autonomy, diversity, response extensiveness, and response efficiency. We identified and solicited 25 target projects from the survey sample and 10 projects agreed to participate in our post hoc case studies.

[2]Software team response extensiveness and software functionality are conceptually distinct constructs in that the former only concerns the extent of team responses to requirement changes that arise during the project whereas the latter concerns meeting the entire requirements including both the original and changed requirements.

[3]Software team response efficiency and on-time completion are conceptually distinct constructs in that the former only concerns the efficiency of team responses to requirement changes that arise during the project whereas the latter concerns on-time completion of the entire project.

[4]Similarly, software team response efficiency and on-budget completion are conceptually distinct constructs.

Measures

The final items for measuring the constructs are shown in Appendix B. We developed new measures for the two agility dimensions as they are newly proposed with no existing measures in the literature. Software team response extensiveness was measured by how much a software team incorporates changing requirements in system scope, input data, output data, business rules/processes, data structure, and user interfaces. Software team response efficiency was measured by the relative level of time, cost, personnel, and resources needed by the software team to respond to and incorporate a given requirement change. Software team response efficiency is considered higher when a team needed less effort to incorporate a given requirement change and vice versa. In the original scale of the measures, a higher number indicates more effort and thus lower response efficiency. For intuitive interpretation of results, we reversed the scale in data analysis so that a higher number indicates higher response efficiency.

Software team autonomy was measured by four items that were adapted from prior literature (Breaugh 1985; Janz et al. 1997; Zmud 1982). These items tapped into the extent to which the software team had discretion, freedom, and independence in making project-related decisions, such as choosing tools/technologies, setting goals, handling user requirement changes, and assigning personnel to the team. Software team diversity was measured by four items adapted from prior literature (Campion et al. 1993). These items tapped into diversity and heterogeneity of team members' expertise areas, skills, prior work experiences, and functional backgrounds.

Software development performance was measured by three dimensions: on-time completion, on-budget completion, and software functionality. Based on the objective data of project start date, planned completion date, and actual completion date, we computed time overrun rate using the ratio of time overrun (i.e., actual completion date minus planned completion date) to planned duration (i.e., planned completion date minus project start date). Greater time overrun rate indicated lower on-time completion performance. Similarly, we obtained planned project cost and actual project cost and computed cost overrun rate using the ratio of cost overrun (i.e., actual project cost minus planned project cost) to planned project cost. Greater cost overrun rate indicated lower on-budget completion performance. In addition to these objective measures, we measured perceived on-time completion and perceived on-budget completion for cross-validation (Deephouse et al. 1996; Nidumolu 1995). Software functionality was measured by four items adapted from prior literature (Nidumolu 1995; Weitzel and Graen 1989). These items measured the extent to which the final software system achieved functional goals, met user requirements, satisfied user needs, and met technical requirements.

Results

Characteristics of the Survey Sample

We received survey responses from the managers of 565 different projects. After eliminating 60 invalid responses, we retained 505 usable responses for data analysis, resulting in an effective response rate of 29.0 percent. As shown in Table 3, the sample represented a wide range of industry sectors and included small, medium, and large organizations with $2.5 billion annual sales and 14,786 employees on average. The sample was also representative of small, medium, and large software development projects. On average, the sample projects had a budget of $2 million, 34 team members, and duration of 12 months. All respondents were project managers of software development projects with their affiliations and functional backgrounds including internal IT managers, internal business managers, and external IT consultants.

To examine the possibility of nonresponse bias, we split the sample into two half-sized subgroups based on the time when each response was received (Bailey 1987; Sivo et al. 2006). We then compared the early response group with the late response group on demographic and project variables such as project duration, team size, project type, industry, organizational size, and PMP (Project Management Professional) certification. No significant differences between the two groups on these variables were found, indicating that nonresponse bias was not likely to be an issue.

To assess whether or not potential common method bias was a significant issue (Malhotra et al. 2006), we performed two different statistical analyses. First, we tested the consistency between objective measures and perception-based Likert-scale measures of two variables: on-time completion and on-budget project completion. We found high correlations between the objective measures and the perception-based measures ($r = .715$ for on-time completion, $r = .762$ for on-budget completion). Second, we conducted a Harman's one-factor test (Podsakoff and Organ 1986) on all of the latent constructs. Results showed that multiple factors are present and the most covariance explained by one factor is only 25.2 percent, indicating that common method biases are not likely to be a serious concern (Podsakoff and Organ 1986).

Table 3. Characteristics of the Survey Sample

Organizations		***Software Projects***	
Industry category		Number of project members	
Consulting	5.8%	Less than 10	24.7%
Finance/Insurance	20.1%	10 – 50	55.7%
Government	8.7%	Over 50	19.6%
Healthcare	5.8%		
Manufacturing	13.9%	Project budget	
Retail	5.1%	Less than $100,000	17.8%
Software	10.1%	$100,000 – $1 million	40.9%
Telecom/Network	5.4%	Over $1 million	41.3%
Transportation	4.0%		
Utility	7.6%	Project duration	
Other	13.5%	Less than 6 months	24.5%
		6 – 12 months	40.5%
Sales		Over 12 months	35.0%
Less than $100 million	26.2%		
$100 million – $1 billion	31.1%	***Respondents***	
Over $1 billion	42.7%	Affiliation/Background	
		Internal IT manager	63.4%
Number of Employees		Internal business manager	9.2%
Less than 1,000	26.8%	External IT consultant	27.4%
1,000 – 10,000	40.6%		
Over 10,000	32.6%		

Measurement Validation

We modeled the indicators of team autonomy, team diversity, on-time completion, on-budget completion, and software functionality as reflective measures. However, we modeled the indicators of response extensiveness and response efficiency as formative measures since these indicators are not expected to have covariation within the same latent construct and they are causes of, rather than caused by, their latent construct (Petter et al. 2007). Reflective indicators and formative indicators require different approaches and criteria for validating reliability, convergent validity, and discriminant validity (Gefen et al. 2000; Petter et al. 2007). Our validation results suggest that all reflective measures demonstrated satisfactory reliability and construct validity and all formative measures demonstrated satisfactory construct validity and no significant multicollinearity. Therefore, all of the measures were valid and reliable. Detailed procedures and results of measurement validation are presented in Appendix C.

Test of the Structural Model

The final sample size for the analysis of the proposed structural model was 399 after excluding 106 responses with missing objective data for project duration and budget. We conducted t-tests to compare these 399 projects with the 106 projects excluded on variables such as firm annual sales, firm employee size, project team size, project type, and project manager's project management experience. No significant differences between the two groups on these variables were found.

Our PLS results are shown in Figure 2. Team autonomy has a significant negative effect (–.272, $p < .01$) on response extensiveness and a significant positive effect (.247, $p < .01$) on response efficiency, supporting H1b and H2. Team diversity has a significant positive effect (.261, $p < .01$) on response extensiveness, supporting H3. However, team diversity does not show a significant effect on response efficiency,

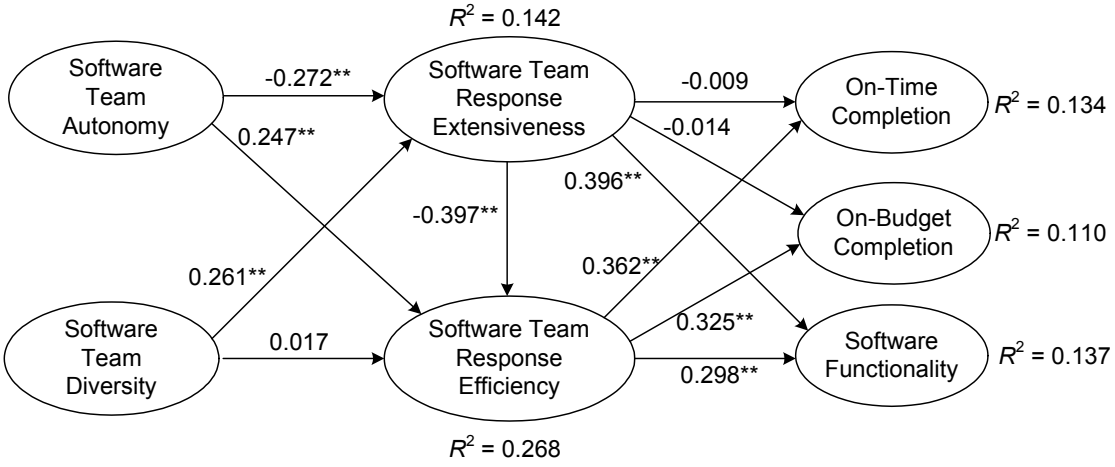

Figure 2. PLS Results

not supporting H4. Response extensiveness has a significant negative effect (–.397, $p < .01$) on response efficiency, thus supporting H5. Response extensiveness has a significant positive affect (.396, $p < .01$) on software functionality, supporting H6c. However, it has no significant effects on either on-time completion or on-budget completion, not supporting H6a or H6b. Response efficiency demonstrates significant positive effects on on-time completion (.362, $p < .01$), on-budget completion (.325, $p < .01$), and software functionality (.298, $p < .01$), supporting H7a, H7b, and H7c.

Response extensiveness and response efficiency collectively explain 13.4 percent of the variance in on-time completion, 11.0 percent in on-budget completion, and 13.7 percent in software functionality. In summary, the results provide support for Hypotheses 1b, 2, 3, 5, 6c, 7a, 7b, and 7c but no support for Hypotheses 1a, 4, 6a, or 6b. In addition, we tested a modified second-order PLS model to examine the effects of response extensiveness and response efficiency on overall software development performance by combining all three performance measures. We found that both response extensiveness and response efficiency have a significant positive effect on the overall development performance. Detailed results are reported in Appendix D.

Although the PLS results support the majority of our hypotheses and reveal interesting findings, more in-depth investigation of individual cases is deemed necessary to explore plausible explanations for unsupported hypotheses, cross-validate supported hypotheses, and provide rich insights for the complex, dynamic relationships among the constructs above and beyond their path coefficients. We discuss below the 10 case studies that we conducted.

Results of Post Hoc Case Studies

As shown in Table 4, these cases represent diverse software development profiles in terms of types of system development, industry sectors, and project size. The cases showed various combinations of different degrees of autonomy, diversity, response extensiveness, response efficiency, and project performance. Most cases used a combination of agile development approaches and traditional waterfall approaches, whereas Case A used an agile development approach alone, Case F used a vendor proprietary evolutionary/iterative approach, and Case H used the waterfall approach.[5] We interviewed the project manager for each project and, when possible, the project manager identified one or two team members for further interviews. In total, we conducted 17 interviews for the 10 cases. Table 5 shows a summary of key findings.

[5]To ensure variability in software development agility among the cases, we selected projects with varying levels of agile methods, ranging from a pure agile approach, to hybrid approaches, to a waterfall approach.

Table 4. Case Description

Case	Project Objective	Duration Cost Team size	Development Methodology	Autonomy/Diversity/ Response Extent/ Response Efficiency (High/Medium/Low)	Project Performance	Interview
A	Off-the-shelf software implementation for core business processes and business intelligence in a dairy foods company	17 months $30M 65 people	Agile	H/H/M/H	1 month overrun; $1.2M cost overrun; 95% of system functionality goals met	project manager
B	In-house development of manufacturing and financial applications in a manufacturing company	24 months $16M 100 people	Agile + Waterfall	M/M/L/M	Cancelled after 18 months with $9M cost-overrun and 5 month time-overrun	project manager
C	Off-the-shelf software implementation in a retail company	12 months $20M 45 people	Agile + Waterfall	M/H/H/M	3 months overrun; $1.5M cost overrun; 75% system functionality goals met	project manager
D	Major application revision and maintenance in an airline company	12 months $1.3M 50 people	Agile + Waterfall	M/M/M/H	4 month time ahead of schedule; $100K cost overrun; 100% system functionality goals met	project manager
E	In-house application development in a transportation company	8 months $1.5M 100 people	Agile + Waterfall	H/M/M/H	3 month time overrun; $150K cost overrun; 95% system functionality goals met	project manager
F	ERP implementation (finance/accounting module) for a federal agency	12 months $30M 160 people	Evolutionary (vendor proprietary)	M/H/L/M	6 month time overrun; $1M cost overrun; system functionality was marginally satisfactory	project manager & one member
G	In-house development of a faculty load management and reporting system for a community college	18 months $400K 3 people	Agile + Waterfall	H/H/H/M	On-time completion; $100K cost overrun; system functionality was very satisfactory	project manager & two members
H	ERP implementation for a military organization	36 months $226M 403 people	Waterfall	L/H/M/L	Significant time and cost overruns; system functionality was largely satisfactory	project manager & one member
I	In-house development of a data processing system for a government agency	48 months $1M 16 people	Agile + Waterfall	M/L/L/M	10 month time overrun; $1.4M cost overrun; system functionality was unsatisfactory	project manager & two members
J	Implementation of an off-the-shelf fund raising system for a radio station	7 months $450K 4 people	Agile + Waterfall	L/M/M/M	On-time completion; $10K cost overrun; system functionality was largely satisfactory	project manager & one member

Table 5. Key Findings from Case Studies

Issues	Findings	Remark
Relationships between software development agility and performance (H6a/b/c & H7a/b/c)	When responding to a requirement change, software teams evaluate its impact on business and time/cost/scope and its technical difficulty. Their response decision is influenced by these tradeoffs. (Cases G, I)	Insights for H6a/b/c
	Time and/or cost constraint is sometimes a main factor for determining the extent of team response to changing requirements. (Cases B, H)	Insights for H6a/b
	High business-impact changes tend to be addressed regardless of constraints. (Case A)	Insights for H6c
	Too much response interrupts project continuity and may result in poor software systems under time/cost pressures. (Cases F, D)	
	Extensive responses to changes in early stages can save time and cost in later stages. (Case G)	Explains unsupported H6a/b
	Pursuit of response efficiency may cause quality/functionality problems. (Case A)	Insights for H7c
	Standardized processes, methodologies, and tools help manage changes, time, and cost. (Case F)	Insights for H6a/b, H7a/b
	Teams re-baseline time line and budget to accommodate large changes and relieve time and cost constraints. (Case H)	
Relationships between response extensiveness and response efficiency (H5)	More responses generally result in lower response efficiency. (Cases C, I)	Validates and explains H5
	Extensive changes affecting the project baseline require approval by upper management, which slows down the response process. (Case C)	
	Too much response results in work overload for the team, which undermines efficiency. (Case E)	
	Responding to too many requirement changes causes lack of focus. (Case D)	
	Clear specification of requirements helps strike the balance. (Case F)	Insights for H5
	Additional requirements are addressed in the next phase or in a separate project. (Cases D, E)	
	Needed changes are spread out across deliverables. (Case E)	
	Effective management of time and cost helps achieve greater agility in both response extensiveness and response efficiency. (Case I)	
Effect of autonomy on agility (H1a/b & H2)	Autonomy generally increases response efficiency. (Case G)	Validates H2
	Autonomy allows software teams to limit their responses to changes in order to meet overall project goals. (Case A)	Validates H1b; explains unsupported H1a
	Autonomy may not be effective for government projects. (Case H)	Insights for H1, H2
	Autonomy may not be effective if team members are not competent. (Case A)	
Effect of diversity on agility (H3 & H4)	Diversity helps solve complex problems effectively. (Cases E, F)	Validates H3
	Diversity helps better translate and understand complex requirement changes. (Case C)	
	Diversity slows down team response due to conflicts and costly communication. (Case D)	Explains unsupported H4
	Diversity can help solve problems quickly. (Case C)	
	Individuals' expertise, knowledge, skills, and mind-set are important for agility. (Case J)	Insights for H3, H4

Relationships Between Software Development Agility and Project Performance

The case study results suggest that software teams dynamically evaluate and manage the complex tensions and tradeoffs between software development agility and software development performance. When responding to a requirement change, software teams holistically assess its business impact, its impact on time/cost/scope goals, and its technical difficulty. Based on these assessments, they determine the extent to which they respond to user requirement changes.

> *It was important to keep balance between how much we respond to user change requests and meeting*

time and cost goals. We probably addressed 80 percent of user requirement requests to make the client happy and satisfied. Yet, at the same time, we tried to minimize less important changes. (Case G)

The stakeholder determined the needed requirements and the nice-to-haves. The software team advises based on technical evaluation what is feasible for each requirement. Change requests are determined during an engineering review board meeting where the project team evaluates the cost and scope of the requirement added to the project. (Case I)

Depending on the specific project context, time is sometimes the main driver for determining how much the team responds to changing requirements, whereas cost can be the main consideration in other cases.

Due to the fact that it was past schedule, I put a very tight and stringent change request policy in place. (Case B)

When the project had a tremendous amount of requirement changes, the project team could not address that many changes simply because they were running out of money. (Case H)

However, we found that high impact, business-disruptive requirement changes tend to be addressed almost always, regardless of time and cost constraints.

We really responded to those things that were truly business disruptive. So we would get the people that needed to be there that knew best about different aspects of that particular problem, both functionally and technically, come up with a design and then build it, regression test it, make sure we understood what the implications were from a usage perspective, and then implement that fix. (Case A)

Although the case results generally confirm the positive effect of response extensiveness on software functionality, when severe time/cost pressures are present, extensive responses may result in poor software functionality due to the lack of task continuity and integration.

Too much response may interrupt the flow and continuity of the project execution. Too much response also has a tendency to create rush changes to meet implementation schedules, which can lead to poorly defined and developed solutions. (Case F)

The tension is you're going to be spread too thin and you're not going to get them tested well enough and integrated well enough, and then you're not going to have a good product at the end. (Case D)

Notably, some cases suggest that extensive responses should not necessarily negatively affect on-time and on-budget completion as we hypothesized. If software teams respond to many important requirement changes in early stages, they can actually save time and cost in later stages. This finding may explain why we did not discover a negative effect of response extensiveness on on-time and on-budget completion with our PLS analysis.

We tried to respond to changes as much as possible early on, and this turned out to save us much time and cost down the road. We did not have to deal with many changes toward the end of the project. (Case G)

Furthermore, our case results suggest what appears to be contradictory to the PLS results: pursuit of response efficiency may cause software functionality problems because efficiency may come at the cost of quality if software teams are not sufficiently competent.

We were pretty efficient about it, but not always effective, because we would make mistakes. People would put the change in and everything would be good but then they forgot that they didn't have the security set up right. So I think the team was pretty efficient but it wasn't quite as effective just because of human error and complexity. (Case A)

While it is a challenge for software teams to be agile while meeting other project goals, we found that standardized processes/methodologies/tools help manage this tension. When possible, teams sometimes re-baseline their project goals and relieve time and cost pressures in order to accommodate large changes.

I found that following a standard methodology is critical to managing scope and changes. An established change control process with tools that track changes are essential for managing CRs (change requests), cost, and schedule. (Case F)

Instead of responding to large changes under the current constraints of time, cost, and scope goals, we sometimes revise the project baseline and set new goals to incorporate large-scale changes. By so doing, we are free from unrealistic constraints

> *that did not account for the emergent changes and can be more flexible in responding to the changes.* (Case H)

Two cases are worth noting: Case B was a big failure, whereas Case D was a remarkable success, among other cases. The difference in their performance can be partly explained by their different levels of response extensiveness and response efficiency as shown in Table 4. Furthermore, our results reveal that the Case B team used a very strict policy in responding to changing user requirements since they were experiencing severe time and cost overruns halfway through the project and because they lacked important skills. The lack of response to important requirement changes rendered the project unsustainable.

> *The knowledge of certain aspects of this new application was something we didn't have in our skill set, which made it difficult to turn things around.* (Case B)

In contrast, the team in Case D was highly efficient in responding to changes and focused on addressing high-priority changes. Although they did not incorporate every single change request, the final system successfully met most of the important user requirements.

Relationships Between Response Extensiveness and Response Efficiency

With respect to the tension between response extensiveness and response efficiency, our case results generally confirm the tradeoff between the two.

> *There is a diminishing rule of return on that. The more change requests you get, you absolutely will suffer on the efficiency and quality of looking at that change.* (Case C)

> *The project team yielded fewer implementations but greater quality and efficiency for each response, as opposed to more responses but less efficiency and quality.* (Case I)

Our case results suggest several plausible explanations for this tradeoff relationship. Responses to extensive requirement changes are more likely to require upper management approval due to their significant impact on business and project goals. Furthermore, too much response results in work overload and lack of focus, thus decreasing response efficiency.

> *I knew that, based on my forecast, making this change would send me so close to budget or over budget that it would switch a dashboard color on the project, and I would actually need management approval for that.* (Case C)

> *There were times where we had a lot of change going on in the project and the team got overwhelmed and we had to push back.* (Case E)

> *If I responded to too many change requests, I not only got stretched too thin and lost focus but also had to shoot at moving targets, and sometimes I felt we didn't even have targets.* (Case D)

We found that project managers tend to believe that they can strike the right balance if user requirements are clearly specified and communicated. Furthermore, by spreading out necessary changes across different phases, deliverables, or projects, software teams may achieve high response efficiency while responding to changes extensively. Effective management of time and cost also helps achieve both dimensions of agility simultaneously.

> *The balance between response extensiveness and efficiency comes from the ability of the team to clearly define and document specifications related to user requirements.* (Case F)

> *We spread out the deliverables and broke it up into three different deliverables, and that was another way to balance the number of changes we had on this project. We could focus on certain changes for the December deliverable, then focus on the changes for the January deliverable, then focus on the final changes.* (Case E)

> *It is possible to reach both response extensiveness and response efficiency if the project cost and time is managed effectively throughout. To do so, the project team needs to leverage the expertise of the staff to make solid project decisions.* (Case I)

Effect of Team Autonomy on Software Development Agility

Our case results confirm that team autonomy generally has a positive effect, mainly on response efficiency because of empowered decision making by team members.

> *Each team member was able to respond to small system changes individually although the whole team*

discussed change requests that are important. We were very efficient in responding to change partly due to our authority to make decisions. (Case G)

We found that autonomous software teams occasionally limited their responses to changing requirements in order to meet project goals because they were empowered to choose if and how to respond to changes, whereas less autonomous teams had no choice but to take orders from users and implement them. This finding partly explains why team autonomy was found to have a negative effect, rather than a positive effect, on response extensiveness.

If you can't put a boundary around that, that's where you go way out of whack on budget and schedule. The business always wants to add additional things, which were never in the scope of what the project was supposed to accomplish. We said "No" to a lot of change requests because we were empowered to make decisions on our own. (Case A)

Effect of Team Diversity on Software Development Agility

Our case results confirm that team diversity improves response extensiveness because it helps solve various problems effectively. Diverse expertise also makes it easier to translate and understand a variety of requirement changes.

The more diverse team will be better able to respond to changes because people will bring different levels of experience, different background, different skill sets. A team that doesn't have that diversity can get tunnel vision on a solution and not be as open to other options. (Case E)

The subject matter experts were actually required to attend and speak to a change so that it could be translated, if you will, for those that don't understand it, which is absolutely critical. (Case C)

We found that diversity can decrease response efficiency due to conflicts and costly communication. On the other hand, however, diversity can increase response efficiency through accelerated problem-solving enabled by readily available expertise and skills. These findings may explain why we found no significant effect of team diversity on response efficiency.

The diversity made it more difficult to communicate and manage change, because the change required interaction amongst a diversity of workgroups, and that made it harder for people to be on the same page and agree to these changes. (Case D)

The make-up of the team, by getting a lot of subject matter experts, resulted in the ability to get answers quickly and to have sort of an automatic trust that the person knows the right answer and you don't have to look for somebody else to validate that. (Case C)

Discussion

Implications for Research and Practice

Our research approach, combining both quantitative and qualitative data analyses, allowed us to not only statistically test the hypotheses, but also complement the quantitative PLS results with richer explanations and insights obtained from the case studies. The benefits from the case studies are threefold: (1) the case study results validate and explain the supported hypotheses; (2) the results provide explanations for the unsupported hypotheses; (3) the results reveal rich, additional insights on the complex relationships among the constructs above and beyond the quantitative results.

The findings and insights of this research have significant implications to research and practice in agile development approaches. As our results suggest a negative effect of response extensiveness on response efficiency, examining only the aggregate agility may produce misleading results. Researchers should distinguish between response extensiveness and response efficiency when developing and testing theories in agile development. This negative effect occurs in part because extensive responses are likely to cause work overload and lack of software team focus and require time-consuming involvement of upper management. However, a possible reverse direction of the relationship—that is, response efficiency affecting response extensiveness—should not be completely ruled out as the relationship is yet to be fully understood. A software team's efficiency in responding to changes may determine the extent of changes to which the team responds. With this reasoning, however, response efficiency is expected to positively affect response extensiveness, which is not consistent with our results. Future research should further investigate if the reverse causality can happen under certain conditions. Practitioners need to be aware of the multifaceted nature of software development agility and understand the tension between its two different dimensions in order to build appropriate types of agility.

Software teams can maintain an appropriate balance between the two agility dimensions by implementing such agile development practices as short, incremental iterations and time boxing methods.

The agile literature tends to take a simplistic view on the tension between software development agility and development performance; it views agility to be universally desirable and does not recognize differential effects of agility on different aspects of development performance. Our PLS results show that response extensiveness has a positive effect only on software functionality, whereas response efficiency has a positive effect on on-time/on-budget completion as well as software functionality. Therefore, software teams should take cautions when implementing such agile principles as "*Welcome changing requirements even late in development*": when time and cost are top priorities, teams can be better off by selectively responding to changing requirements and thus increasing response efficiency. Agile practices such as "*decision in one hour*" in Scrum can be useful for improving response efficiency.

Our case results provide a plausible explanation for the nonsignificant effect of response extensiveness on on-time and on-budget completion. While extensive responses generally require substantial time, cost, and resources, extensive responses in early development stages can save software development time and cost for later development stages. Therefore, extensive responses might have both positive and negative effects on on-time and on-budget completion, resulting in a nonsignificant net effect. Our case results also suggest that, contrary to our PLS results, it is possible for response efficiency to negatively affect software functionality as efficiency might compromise system quality if the team is not competent. Similarly, extensive responses can negatively affect software functionality under severe time/cost constraints.

Agile development approaches advocate self-organizing, autonomous teams with cross-functional, diverse members. However, no theoretical foundation or empirical evidence has been provided in support of the principle. This research provides empirical evidence that team autonomy and team diversity are important team variables that organizations can control to build their software development agility. More importantly, team autonomy and team diversity do not universally increase software development agility. The results suggest that increased autonomy without increased diversity may result in decreased response extensiveness, and that only autonomy, not diversity, increases response efficiency. The results suggest that a software team needs to manage team autonomy and diversity based on which dimension of agility the team needs to address. Software teams should be aware of a potential negative effect of self-organization on their ability to respond to a wide range of changes. When implementing "*the whole team*" and "*pair programming*" with XP, the team and the pairs should have a high degree of diversity for greater response extensiveness.

Our case results suggest that autonomous software teams may strictly limit their responses to changing requirements in order to meet other project goals such as time and cost. However, researchers should not entirely rule out the possibility that autonomy may actually increase response extensiveness since autonomous teams, in theory, are free to choose their actions in either direction. Future research should further investigate this relationship. In addition, our case results suggest two opposite effects of team diversity on response efficiency. While diversity may cause conflicts and costly communication, it may accelerate a software team's response process, thanks to readily available expertise and skills. These findings partly explain the nonsignificant effect of diversity on response efficiency as its positive and negative effects may be cancelled out.

Limitations

Although we employed a rigorous, multiphase approach to the development of new measures for response extensiveness and response efficiency, the new measures have some limitations. While response extensiveness is measured by the count of incorporated changes vis-à-vis total changes, it would have been more accurately measured using the amount of additional scope, above and beyond the original scope, in terms of such metrics as function points. Furthermore, response extensiveness measures the extent to which a software team actually implemented and incorporated various changes into the system while not fully capturing other forms of team responses. A software team may just as well respond to requirement changes by rejecting or postponing them to meet other important project goals. A more comprehensive measure is desirable to fully capture different forms of software team responses.

Our measure for response efficiency did not take into account the fact that different requirement changes from different development phases might have different levels of business impact and response difficulties. Therefore, combining all phases in the measure may not precisely capture how efficient a team's responses were. To measure it more accurately, appropriate weights based on relative impact and difficulty need to be assigned to different types of changes. Another

issue associated with the response efficiency measure is that survey respondents may have had a lack of clear guidance in interpreting the measurement question and assessing relative time, cost, and resources required in responding to a given requirement change. Although we have found no statistical evidence that suggests any measurement problems due to potential misinterpretation, future research is needed to further validate the measure.

Conclusions

Due to the ever-increasing uncertainty in business and technology environments, the agility to effectively deal with changing requirements has become an imperative, not an option, for software development. Given the complex relationships among response extensiveness, response efficiency, team autonomy, team diversity, and software development performance, software teams face difficult challenges in identifying and achieving the right balance between the two agility dimensions. While the prior agile development literature offers little guidance for such challenges, this research offers some useful insights that are theoretically based and empirically tested. Software teams should first prioritize the performance goals of time, cost, and functionality, which will determine how much each dimension of agility is needed. That, in turn, will determine how much autonomy and diversity their software teams should require.

Since this research is one of the initial efforts to empirically examine the principles and practices of agile development approaches, we believe that many important questions and issues are yet to be answered in this important area. We hope our study serves as a stepping-stone for developing and testing theories that guide the agile development principles and practices so that organizations can effectively build and sustain software development agility that will ultimately improve their software development performance.

Acknowledgments

This study was supported by research grants provided by the University of Minnesota, the Juran Center for Leadership in Quality, and the UPS (United Parcel Service) Foundation. The Information Systems Specific Interest Group of the Project Management Institute sponsored the collection of the survey data. We thank Carl Adams, Izak Benbasat, Erran Carmel, Gordon Davis, William DeLone, J. Alberto Espinosa, Jungpil Hahn, Anita LaSalle, Bob Zmud, and research workshop participants at the American University, Lehigh University, Syracuse University, Tulane University, the University of British Columbia, and the University of Minnesota for helpful comments on earlier versions of the paper.

References

Agile Alliance. 2001. "Manifesto for Agile Software Development" (www.agilemanifesto.org).

Aladwani, A. M. 2002. "An Integrated Performance Model of Information Systems Projects," *Journal of Management Information Systems* (19:1), pp. 185-210.

Ancona, D., and Caldwell, D. 1992. "Demography and Design: Predictors of New Product Team Performance," *Organization Science* (3:3), pp. 321-341.

Ashby, W. R. 1956. *An Introduction to Cybernetics*, London: Chapman and Hall.

Austin, R., and Devin, L. 2003. *Artful Making: What Managers Need to Know About How Artists Work*, Boston: Financial Times Prentice Hall.

Bailey, K. D. 1987l *Methods of Social Research*, New York: Free Press.

Baskerville, R. L. 2006. "Artful Planning," *European Journal of Information Systems* (15:2), pp. 113-115.

Beck, K., and Andres, C. 2005. *Extreme Programming Explained: Embrace Change*, Boston: Addison-Wesley.

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

Breaugh, J. A. 1985. "The Measurement of Work Autonomy," *Human Relations* (38:6), pp. 551-570.

Campion, M. A., Medsker, G. J., and Higgs, A. C. 1993. "Relations Between Work Group Characteristics and Effectiveness: Implications for Designing Effective Work Groups," *Personnel Psychology* (46:4), pp. 823-850.

Chin, W. W. 1998. "The Partial Least Square Approach to Structural Equation Modeling," in *Modern Methods for Business Research*, G. A. Marcoulides (ed.), Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

Chow, T., and Cao, D.-B. 2008. "A Survey Study of Critical Success Factors in Agile Software Projects," *The Journal of Systems and Software* (81:6), pp. 961-971.

Clark, K. B., and Fujimoto, T. 1991. *Product Development Performance*, Boston: Harvard Business School Press.

Coad, P., De Luca, J., and Lefebre, E. 1999. *Java Modeling in Color*, Englewood Cliffs, NJ: Prentice Hall.

Cockburn, A. 2001. *Agile Software Development*, Boston: Addison-Wesley.

Cockburn, A. 2007. *Agile Software Development: The Cooperative Game*, Boston: Addison-Wesley.

Cockburn, A., and Highsmith, J. 2001. "Agile Software Development: The People Factor," *IEEE Computer* (34:11), pp. 131-133.

Conboy, K., and Fitzgerald, B. 2004. "Toward a Conceptual Framework of Agile Methods," in *Proceedings of the 2004 ACM Workshop on Interdisciplinary Software Engineering Research,* Newport Beach, CA, November 5, 2004, pp. 37-44.

Deephouse, C., Mukhopadhyay, T., Goldenson, D. R., and Keller, M. I. 1996. "Software Processes and Project Performance," *Journal of Management Information Systems* (12:3), pp. 187-205.

Diamantopoulos, A., and Siguaw, J. A. 2006. "Formative Versus Reflective Indicators in Organizational Measure Development:

A Comparison and Empirical Illustration," *British Journal of Management* (17:4), pp. 263-282.

Diamantopoulos, A., and Winklhofer, H. M. 2001. "Index Construction with Formative Indicators: An Alternative to Scale Development," *Journal of Marketing Research* (38:2), pp. 269-277.

Dybå, T., and Dingsøyr, T. 2008. "Empirical Studies of Agile Software Development: A Systematic Review," *Information and Software Technology* (50:9-10), pp. 833-859.

Erickson, J., Lyytinen, K., and Siau, K. 2005. "Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research," *Journal of Database Management* (16:4), pp. 88-100.

Fitzgerald, B., Hartnett, G., and Conboy, K. 2006. "Customising Agile Methods to Software Practices at Intel Shannon," *European Journal of Information Systems* (15:2), pp. 200-213.

Fruhling, A., and De Vreede, G.-J. 2006. "Field Experiences with eXtreme Programming: Developing an Emergency Response System," *Journal of Management Information Systems* (22:4), pp. 39-68.

Gefen, D., and Ridings, C. M. 2002. "Implementation Team Responsiveness and User Evaluation of Customer Relationship Management: A Quasi-Experimental Design Study of Social Exchange Theory," *Journal of Management Information Systems* (19:1), pp. 47-69.

Gefen, D., Straub, D. W., and Boudreau, M.-C. 2000. "Structural Equation Modeling and Regression: Guidelines for Research Practice," *Communications of the AIS* (4:7), pp. 1-77.

Henderson-Sellers, B., and Serour, M. K. 2005. "Creating a Dual-Agility Method: The Value of Method Engineering," *Journal of Database Management* (16:4), pp. 1-23.

Highsmith, J. 2000. *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, New York: Dorset House Publishing.

Highsmith, J. 2002. "Agile Project Management: Principles and Tools," *Cutter Consortium Executive Report*, March 9 (available online at http://www.cutterconsortium.com/research/2004/edge040309.html).

Highsmith, J. 2004. *Agile Project Management*, Boston: Addison-Wesley.

Imai, K., Ikujiro, N., and Takeuchi, H. 1985. "Managing the New Product Development Process: How Japanese Companies Learn and Unlearn," in *The Uneasy Alliance*, R. H. Hayes, K. B. Clark, and C. Lorenz (eds.), Boston: Harvard Business School Press, pp. 337-375.

Janz, B. D., Wetherbe, J. C., Davis, G. B., and Noe, R. A. 1997. "Reengineering the Systems Development Process: The Link between Autonomous Teams and Business Process Outcomes," *Journal of Management Information Systems* (14:1), pp. 41-68.

Kaplan, B., and Duchon, D. 1988. "Combining Qualitative and Quantitative Methods in Information Systems Research: A Case Study" *MIS Quarterly* (12:4), pp. 571-586.

Kelly, A. 2008. *Changing Software Development: Learning to Become Agile*, Chichester, England: John Wiley & Sons.

Kerzner, H. 2005. *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*, Hoboken, NJ: Wiley.

Klimoski, R. J., and Mohammed, S. 1994. "Team Mental Model: Construct or Metaphor," *Journal of Management* (20:2), pp. 403-437.

Koch, C. 2006. "The Truth about SOA," *CIO*, June 15 (available online at http://www.cio.com/article/21975/The_Truth_About_SOA).

Larman, C. 2004. *Agile & Iterative Development: A Manager's Guide*, Boston: Addison-Wesley.

Lee, A. S. 1991. "Integrating Positivist and Interpretive Approaches to Organizational Research," *Organization Science* (2:4), pp. 342-365.

Loch, K. D., Straub, D. W., and Kamel, S. 2003. "Diffusing the Internet in the Arab World: The Role of Social Norms and Technological Culturation," *IEEE Transactions on Engineering Management* (50:1), pp. 45-63.

Lyytinen, K., and Rose, G. M. 2006. "Information System Development Agility as Organizational Learning," *European Journal of Information Systems* (15:2), pp. 183-199.

MacCormack, A., Verganti, R., and Iansiti, M. 2001. "Developing Products on 'Internet Time': The Anatomy of a Flexible Development Process," *Management Science* (47:1), pp. 133-150.

Malhotra, N. K., Kim, S. S., and Patil, A. 2006. "Common Method Variance in IS Research: A Comparison of Alternative Approaches and a Reanalysis of Past Research," *Management Science* (52:12), pp. 1865-1883.

Mathieu, J., Goodwin, G. F., Heffner, T. S., Salas, E., and Cannon-Bowers, J. A. 2000. "The Influence of Shared Mental Models on Team Process and Performance," *Journal of Applied Psychology* (85:2), pp. 273-283.

McGrath, R. G. 2001. "Exploratory Learning, Innovative Capacity, and Managerial Oversight," *Academy of Management Journal* (44:1), pp. 118-131.

Miller, C. C., Burkle, L. M., and Glick, W. H. 1998. "Cognitive Diversity Among Upper-Echelon Executives: Implications for Strategic Decision Processes," *Strategic Management Journal* (19:1), pp. 39-58.

Milliken, F. J., and Martins, L. L. 1996. "Searching for Common Threads: Understanding the Multiple Effects of Diversity in Organizational Groups," *Academy of Management Review* (21:2), pp. 402-433.

Mingers, J. 2001. "Combining IS Research Methods: Towards a Pluralist Methodology," *Information Systems Research* (12:3), pp. 240-259.

Mitchell, V. L. 2006. "Knowledge Integration and Information Technology Project Performance," *MIS Quarterly* (30:4), pp. 919-939.

Moe, N. B., Dingsøyr, T., and Dybå, T. 2008. "Understanding Self-Organizing Teams in Agile Software Development," *Proceedings of the 19th Australian Conferences on Software Engineering*, Perth, Australia, pp. 76-85.

Moore, G. C., and Benbasat, I. 1991. "Development of an Instrument to Measure the Perceptions of Adopting an Information Technology Innovation," *Information Systems Research* (2:3), pp. 192-222.

Nerur, S., and Balijepally, V. 2007. "Theoretical Reflections on Agile Development Methodologies," *Communications of the ACM* (50:3), pp. 79-83.

Nidumolu, S. R. 1995. "The Effect of Coordination and Uncertainty on Software Project Performance: Residual Performance Risk as an Intervening Variable," *Information Systems Research* (6:3), pp. 191-219.

Pelled, L. H., Eisenhardt, K. M., and Xin, K. R. 1999. "Exploring the Black Box: An Analysis of Work Group Diversity, Conflict, and Performance," *Administrative Science Quarterly* (44:1), pp. 1-28.

Petter, S., Straub, D., and Rai, A. 2007. "Specifying Formative Constructs in Information Systems Research," *MIS Quarterly* (31:4), pp. 623-656.

Podsakoff, P. M., and Organ, D. W. 1986. "Self-Reports in Organizational Research: Problems and Prospects," *Journal of Management* (12:4), pp. 531-544.

Qumer, A., and Henderson-Sellers, B. 2008. "An Evaluation of the Degree of Agility in Six Agile Methods and its Applicability for Method Engineering," *Information and Software Technology* (50:4), pp. 280-295.

Schmidt, R., Lyytinen, K., Keil, M., and Cule, P. 2001. "Identifying Software Project Risks: An International Delphi Study," *Journal of Management Information Systems* (17:4), pp. 5-36.

Schwaber, K., and Beedle, M. 2002. *Agile Software Development with Scrum*, Upper Saddle River, NJ: Prentice-Hall.

Sharp, H., and Robinson, H. 2004. "An Ethnographic Study of XP Practice," *Empirical Software Engineering* (9:4), pp. 353-375.

Sivo, S. A., Saunders, C., Chang, Q., and Jiang, J. J. 2006. "How Low Should You Go? Low Response Rates and the Validity of Inference in IS Questionnaire Research," *Journal of the AIS* (7:6), pp. 351-414.

Stapleton, J. 1997. *DSDM: Dynamic Systems Development Method*, Harlow, England: Addison Wesley.

Tata, J., and Prasad, S. 2004. "Team Self-Management, Organizational Structure, and Judgments of Team Effectiveness," *Journal of Managerial Issues* (16:2), pp. 248-265.

Tatikonda, M. V., and Rosenthal, S. R. 2000. "Successful Execution of Product Development Projects: Balancing Firmness and Flexibility in the Innovation Process," *Journal of Operations Management* (18:4), pp. 401-425.

Trauth, E. M., and Jessup, L. M. 2000. "Understanding Computer-Mediated Discussions: Positivist and Interpretive Analyses of Group Support System Use," *MIS Quarterly* (24:1), pp. 43-79.

Turner, J. C., Hogg, M. A., Oakes, P. J., Reicher, S. D., and Wetherell, M. S. 1987. *Rediscovering the Social Group: A Self-Organization Theory*, New York: Blackwell.

Tushman, M. L., and O'Reilly, C. A. 1996. "Ambidextrous Organizations: Managing Evolutionary and Revolutionary Change," *California Management Review* (38:4), pp. 8-30.

Van de Ven, A. H., and Delbecq, A. L. 1974. "The Effectiveness of Nominal, Delphi, and Interacting Group Decision Making Processes," *Academy of Management Journal* (17:4), pp. 605-621.

Watson, E. W., Kumar, K., and Michaelsen, L. K. 1993. "Cultural Diversity's Impact on Interaction Process and Performance: Comparing Homogeneous and Diverse Take Groups," *Academy of Management Journal* (36:3), pp. 590-602.

Webber, S. S., and Donahue, L. M. 2001. "Impact of Highly and Less Job-Related Diversity on Work Group Cohesion and Performance: A Meta-Analysis," *Journal of Management* (27:2), pp. 141-162.

Weitzel, J. R., and Graen, G. B. 1989. "Systems Development Project Effectiveness: Problem-Solving Competence as a Moderator Variable," *Decision Sciences* (20:3), pp. 507-531.

Werts, C. E., Linn, R. L., and Joreskog, K. G. 1974. "Intraclass Reliability Estimates: Testing Structural Assumptions," *Educational and Psychological Measurement* (34:1), pp. 25-33.

Williams, K. Y., and O'Reilly, C. A. 1998. "Demography and Diversity in Organizations: A Review of 40 Years of Research," in *Research in Organizational Behavior*, B. M. Staw and L. L. Cummings (eds.), Greenwich, CT: JAI Press, pp. 77-140.

Yi, M. Y., and Davis, F. D. 2003. "Developing and Validating an Observational Learning Model of Computer Software Training and Skill Acquisition," *Information Systems Research* (14:2), pp. 146-169.

Zmud, R. W. 1982. "Diffusion of Modern Software Practices: Influence of Centralization and Formalization," *Management Science* (28:12), pp. 1421-1431.

About the Authors

Gwanhoo Lee is an associate professor and UPS Faculty Scholar in the Kogod School of Business at the American University, Washington, DC. He is the director of the Center for IT and the Global Economy at the American University. His research areas include software development agility and complexity, distributed software teams, IT-enabled collaboration and innovation, technology adoption, and CIO leadership. He has been working closely with IT executives from large U.S. organizations on those research areas through collaborative forums and programs. His research has been published in *Journal of Management Information Systems, European Journal of Information Systems, Communications of the ACM, Information & Management, Information Technology and People, IEEE Pervasive Computing, Journal of Information Technology Management,* and *Academy of Management Best Paper Proceedings*, as well as in the proceedings of the International Conference on Information Systems, the Hawaii International Conference on System Sciences, and the Americas Conference on Information Systems. He earned his doctorate in management information systems from the University of Minnesota.

Weidong Xia is a faculty member in the College of Business Administration at Florida International University. His research focuses on IT strategy and governance, software development complexity and flexibility, and innovation adoption. He was on the faculty of the Carlson School of Management at the University of Minnesota. He has worked with a number of large companies as the cofounder and codirector of the CIO Research Consortium. His research has been published in *MIS Quarterly*, *Decision Sciences*, *Journal of Management Information Systems*, *Communications of the ACM*, *European Journal of Information Systems*, *Journal of Information Technology Management, Journal of Statistics and Management Systems, International Journal of Career Development,* and *Journal of End-User Computing*. He received his doctorate in information systems from the University of Pittsburgh.

Appendix A

The Research Process

Table A1 provides detailed information about our five-phase research process, including subject characteristics, outcomes, and techniques/methods/processes. In Phase 1, we formulated research questions and identified key constructs of the research through preliminary field interviews with 36 IS managers and executives who were affiliated with an academic research center at a large U.S. public university. The questions asked during 90-minute semi-structured interviews included

(1) How would you define agility in the context of business software development?
(2) What are the important dimensions of software development agility?
(3) How does software development agility affect software development performance?
(4) What factors are important for building software development agility?

Then, we conducted a focus group brainstorming session with another 45 IS managers who enrolled in the part-time MBA programs at the same university. A brief summary of the preliminary interview results was presented to the group to provide the initial conceptualization of software development agility. Using the nominal group technique (Van de Ven and Delbecq 1974), the participants created a list of measures of software development agility. As a result, we obtained 24 initial measures for software team response extensiveness and software team response efficiency, respectively.

The initial measurement items were refined through a sorting procedure and two pilot tests of the survey instrument. We followed the sorting procedure used by Moore and Benbasat (1991) to qualitatively assess the construct validity of measurement items. Each measurement item was placed into an appropriate category by four judges who were doctoral candidates with extensive IS project experience. The item placement hit ratio was 91 percent, which was calculated by dividing the number of items correctly placed by the total number of items. Several items were dropped because they were too ambiguous or unclear.

The items were further refined by two pilot tests of the survey instrument. The first pilot test was conducted through one-hour interviews with another four IS managers and three IS doctoral students affiliated with the university. The participants evaluated the importance and relevance of each measurement item of software development agility. The items that had higher mean scores in both importance and relevance were retained. As a result, we retained six items for response extensiveness and another six items for response efficiency. The second pilot test involved another 15 IS managers. The participants validated a prototype of our on-line survey questionnaire in terms of readability, format, and wording. Based on the feedback, the format of the questionnaire was improved and the wordings of the measurement items were fine-tuned. There was no overlapping of study participants across the different sessions in Phase 1 and none of them participated in the main field survey in Phase 2.

In Phase 2, we used a Web-based online instrument to collect quantitative survey data from project managers of software development projects. A PMI-ISSIG-sponsored e-mail letter with a hyperlink to our online survey was sent to the target group. The participants were entered into a drawing to receive 10 awards of a PMI-ISSIG official shirt and 40 awards of a $25 gift certificate. A reminder was sent 2 weeks after the initial PMI-ISSIG-sponsored e-mail was sent out, followed by a second email reminder 2 weeks later.

After the data was obtained, we validated convergent validity, discriminant validity, and reliability of the measures in Phase 3, using different approaches for reflective measures and formative measures, following the guidelines recommended by Petter et al. (2007). Appendix C shows the details of measurement validation. In addition, we assessed nonresponse bias and common method variance to ensure that the PLS results were not biased.

In Phase 4, we used partial least square (PLS) to test the research model. PLS is more appropriate than LISREL for exploratory research (Chin 1998; Petter et al. 2007). Response extensiveness and response efficiency are formative latent variables. Furthermore, as these two constructs are newly proposed in this research, the hypotheses are exploratory in nature. After testing the proposed research model, we also tested a modified PLS model where overall software development performance is modeled as a second-order construct that combines on-time completion, on-budget completion, and software functionality.

In Phase 5, we conducted post hoc case studies on 10 software development projects selected from the survey sample in order to provide complementary, additional, richer insights and findings on the complex relationships between software development agility, autonomy, diver-

Table A1. Multiphase, Multimethod, Research Process

Phase	Research Process and Results
Phase 1 Preliminary field studies for problem formulation, construct identification, and measurement development	• Preliminary field interviews – 90 minute semi-structured interviews with 36 IS managers and executives – The interviewees were affiliated with a research center at a large U.S. public university and represented a variety of industries including manufacturing, financial services, healthcare, consulting service, restaurant, insurance, agriculture, transportation, package delivery service, and medical equipment – The interviewees had over 10 years of IS work experience on average – Constructs were identified and defined; research questions were formulated • Focus group session – A one-hour brainstorming session with 45 IS managers – The participants were part-time MBA students at a large U.S. public university – The participants had minimum 3 years of work experience – Using the nominal group technique (Van de Ven and Delbecq 1974), participants created a list of measures of software development agility and ranked them – 24 initial items for response extensiveness and response efficiency were generated • A sorting procedure – The procedure used by Moore and Benbasat (1991) was conducted – Four judges were doctoral candidates in information systems with an average 8 years of prior work experience – A placement hit ratio of 91% • Pilot tests of the survey instrument – The first pilot test involved four IS managers and three doctoral students and evaluated importance and relevance of measures – The second pilot test involved 15 IS managers and validated the on-line survey questionnaire – All participants had at least 4 years of work experience in the IS field – Six items for response extensiveness and response efficiency were retained – The on-line survey questionnaire was finalized
Phase 2 Survey data collection	• An online survey was administered – Target respondents were 1,740 PMI-ISSIG members – 505 valid responses with an effective response ratio of 29%
Phase 3 Measurement validation	• Different methods were used for validating reflective vs. formative measures • Validation of convergent/discriminant validity and reliability • Assessment of nonresponse bias and common method variance
Phase 4 Hypotheses testing	• PLS was used to analyze the final sample of 399 projects • Estimation of structural path coefficients and R^2 • Additional data analysis was conducted • A model with a second order construct for software development performance
Phase 5 Post hoc case studies for rich, additional insights	• Field interviews with 17 project managers and team members from 10 cases • 90 minute semi-structured interviews • Interview results were compared within and across cases • Validation of and plausible explanations for the PLS results • Rich, additional insights and findings • Identification of other important factors for further research

sity, and performance. We conducted 17 semi-structured interviews for 10 cases with project managers and team members (see Table 4 for details). All interviews were one-on-one meetings and were about 90 minutes long. The questions asked during the interview included

- How does the software team determine the importance/priority of changes? How does the software team decide on how a particular change request should be handled?
- How would you describe the relationship and tension between response extensiveness and response efficiency? How does the software team strike a balance between them?

- How did various agile development practices affect software development agility?
- How would you describe the tension between (1) needs for meeting time, cost, and scope, and (2) needs for implementing user requirement changes? How does the software team go about managing this tension?
- How does team autonomy and diversity affect software development agility?
- What are the other organizational/technological factors that affect software development agility?

Each interview was recorded and transcribed for analysis. We identified important comments from each interview and compared them within and across cases. We found that comments from multiple interviewees within cases were largely consistent. However, when there were notable differences, we asked the interviewees to clarify and resolve the differences. We obtained key insights through several iterations of comparing and contrasting interview comments.

Appendix B

Measurement Scales and Items

Software team response extensiveness (formative) (1 = 0%; 2 = 20%, 3 = 40%, 4 = 60%, 5 = 80%, 6 = 100%)
To what extent did the software team actually incorporate requirement changes in each of the following categories? (For example, if the project actually incorporated four out of ten different changes in a specific category, your answer would be 40 %.)

1. System scope (EXT1)
2. System input data (EXT2)
3. System output data (EXT3)
4. Business rules/processes (EXT4)
5. Data structure (EXT5)
6. User interface (EXT6)

Software team response efficiency[1] (formative) (1 = very little; 7 = very much)
How much additional effort was required by the software team to incorporate the following changes? (Effort includes time, cost, personnel, and resources.)

1. System scope (EFF1)
2. System input data (EFF2)
2. System output data (EFF3)
4. Business rules/processes (EFF4)
5. Data structure (EFF5)
6. User interface (EFF6)

Software team autonomy (reflective) (1 = strongly disagree; 7 = strongly agree)

1. The project team was allowed to freely choose tools and technologies (AUTO1)
2. The project team had control over what they were supposed to accomplish (AUTO2)
3. The project team was granted autonomy on how to handle user requirement changes (AUTO3)
4. The project team was free to assign personnel to the project (AUTO4)

Software team diversity (reflective) (1 = strongly disagree; 7 = strongly agree)

1. The members of the project team were from different areas of expertise (DIV1)
2. The members of the project team had skills that complemented each other (DIV2)
3. The members of the project team had a variety of different experiences (DIV3)
4. The members of the project team varied in functional backgrounds (DIV4)

Software functionality (reflective) (1 = strongly disagree; 7 = strongly agree)

1. The software delivered by the project achieved its functional goals (FUNC1)
2. The software delivered by the project met end-user requirements (FUNC2)
3. The capabilities of the software fit end-user needs (FUNC3)
4. The software met technical requirements (FUNC4)

On-time completion (OnTime) (reflective)
1. Objective measure: project start date: (mm/dd/yyyy), planned completion date: (mm/dd/yyyy), actual completion date (mm/dd/yyyy)
2. Perception-based measure[2] (1 = strongly disagree; 7 = strongly agree)
   The project was completed late according to the original schedule

On-budget completion (OnBudget) (reflective)
1. Objective measure: planned project cost: (in dollar), actual project cost: (in dollar)
2. Perception-based measure[2] (1 = strongly disagree; 7 = strongly agree)
   The project was completed over budget according to the original budget

Notes: 1. These scale items were reversed for data analysis. With the reversed scales, higher scores indicated higher response efficiency.
2. These items were reversed for data analysis. With the reversed scales, higher scores indicated higher project performance.

Appendix C

Measurement Validation

The reliability indexes of latent constructs with reflective indicators were evaluated by composite reliability. Composite reliability of 0.70 or higher is considered acceptable (Werts et al. 1974). For adequate convergent and discriminant validity, the square root of the average variance extracted (AVE) should be at least 0.707 and exceed the correlations between the focal construct and other constructs (Gefen et al. 2000). Furthermore, standardized item loadings should be greater than 0.70 and items should load more highly on their intended construct than on other constructs (Gefen et al. 2000).

The measurement validation results for the reflective constructs are shown in Tables C1 and C2. The results suggest that composite reliability indexes for the three perceptually measured reflective constructs (team autonomy, team diversity, and software functionality) are much higher than 0.70. The square root of the variance extracted (AVE) for all constructs is higher than 0.707, all standardized item loadings except for AUTO4 are greater than 0.70, and all items load more highly on their intended construct than on other constructs. Since on-time completion and on-budget completion were each measured by a single item derived from objective performance data, their AVE is 1.0 and their reliability cannot be calculated. Although the factor loading of AUTO4 to autonomy is slightly lower (0.690) than the criterion of .70, this item loads much more highly on autonomy than on other constructs. Moreover, the composite reliability of the latent construct team autonomy decreases if AUTO4 is removed. Therefore, it deems reasonable to retain AUTO4 for data analysis.

Table C1. Reliability and Convergent and Discriminant Validity for Reflective Constructs

Latent Construct	Composite Reliability	Autonomy	Diversity	Functionality	On Time	On Budget
Autonomy	0.839	**0.753**				
Diversity	0.883	0.003	**0.809**			
Functionality	0.966	0.037	0.197	**0.937**		
On Time	n/a	0.269	0.074	0.044	**1.000**	
On Budget	n/a	0.310	0.022	0.029	0.633	**1.000**

Notes: Composite reliability (ρ_c) = $(\Sigma \lambda_i)^2 / [(\Sigma \lambda_i)^2 + \Sigma \text{var}(\varepsilon_i)]$, where λ_i is the component loading to an indicator and var (ε_i) = $1 - \lambda_i^2$; diagonal elements in bold case are the square root of average variance extracted (AVE) by latent constructs from their indicators; off-diagonal elements are correlations among latent constructs.

Table C2. PLS Component-Based Analysis: Cross-Loadings for Reflective Constructs					
Scale Items	**Autonomy**	**Diversity**	**Functionality**	**On Time**	**On Budget**
AUTO1	**0.728**	-0.054	-0.033	0.161	0.204
AUTO2	**0.793**	0.040	0.084	0.304	0.289
AUTO3	**0.802**	0.005	0.052	0.192	0.271
AUTO4	**0.690**	0.028	0.005	0.144	0.167
DIV1	-0.083	**0.758**	0.037	0.017	-0.009
DIV2	0.117	**0.743**	0.297	0.188	0.084
DIV3	-0.005	**0.864**	0.177	0.059	0.006
DIV4	0.007	**0.882**	0.158	0.007	0.006
FUNC1	0.037	0.168	**0.945**	0.053	0.024
FUNC2	0.040	0.190	**0.956**	0.061	0.032
FUNC3	0.054	0.195	**0.943**	0.024	0.019
FUNC4	0.007	0.201	**0.925**	0.030	0.032
On Time	0.267	0.078	0.045	**1.000**	0.633
On Budget	0.310	0.024	0.028	0.633	**1.000**

Notes: To calculate cross-loadings, a factor score for each construct was calculated based on the weighted sum, provided by PLS results. Factor scores were correlated with individual items to calculate cross loadings. Boldface numbers are loadings (correlations) of indicators to their own construct.

We followed the guidelines proposed by Petter et al. (2007) to validate the measurement of software team response extensiveness and software team response efficiency. We used a modified MTMM (multitrait–multimethod matrix) analysis proposed by Loch et al. (2003) for validating convergent and discriminant validity. We created a weighted score for each construct using the formative weights provided by PLS results. We then created a correlation matrix consisting of the indicators and formative latent constructs. If the majority of inter-item correlations and item-to-construct correlations for a given latent construct are significant, the measures achieve convergent validity. If the items tend to correlate more with one another within the same construct than with items of other constructs, the measures achieve discriminant validity. The presence of violation, however, does not necessarily suggest that the formative construct does not have construct validity, because formative indicators do not necessarily have high correlations among them (Petter et al. 2007). If there are violations in the modified MTMM matrix, efforts should be made to understand why these violations occurred. The results shown in Table C3 suggest that all inter-item correlations and item-to-construct correlations for the measures used to assess response extensiveness and response efficiency are significant. They also suggest that all items correlate with one another within the same construct much higher than with items of the other construct, with no exception. Therefore, we concluded that the two formative constructs, response extensiveness and response efficiency, exhibited adequate convergent and discriminant validity.

Assessing reliability is more difficult with formative measures than with reflective measures and it is not always possible to accomplish it (Diamantopoulos and Winklhofer 2001; Petter et al. 2007). In a sense, very high reliability can be undesirable for formative constructs because excessive multicollinearity among formative indicators can destabilize the model (Petter et al. 2007). To ensure that multicollinearity is not a significant issue, we assessed the VIF (variance inflator factor) statistic. If the VIF statistic is greater than 3.3, the conflicting item should be removed as long as the overall content validity of the construct measures is not compromised (Diamantopoulos and Siguaw, 2006). The VIF estimates for the measures of response extensiveness and response efficiency are shown in Table C4. The results suggest that all indicators except for EXT3 have VIF statistics lower than 3.3. The VIF statistic for EXT3 (software team response extensiveness to output data change) is slightly higher (3.83) than the criterion. However, removing this indicator appears to compromise the content validity because its counterpart item EXT2 (software team response extensiveness to input data change) and EFF3 (software team response efficiency to output data change) are integral parts of the data analysis. Thus, removing EXT3 would result in unbalanced content coverage for the response extensiveness and response efficiency constructs. The threshold value for VIF (3.3) in this research is much more conservative compared to traditional criteria such as 5 or 10. Based on the above considerations, we retained the item EXT3 for data analysis.

Table C3. Inter-Item and Item-to-Construct Correlation Matrix for Formative Indicators

	EXT1	EXT2	EXT3	EXT4	EXT5	EXT6	EXT	EFF1	EFF2	EFF3	EFF4	EFF5	EFF6	EFF
EXT1	1.000													
EXT2	0.624	1.000												
EXT3	0.656	0.786	1.000											
EXT4	0.660	0.696	0.747	1.000										
EXT5	0.592	0.660	0.632	0.661	1.000									
EXT6	0.612	0.617	0.669	0.619	0.580	1.000								
EXT	0.896	0.779	0.839	0.905	0.762	0.710	1.000							
EFF1	-0.362	-0.254	-0.271	-0.309	-0.268	-0.239	-0.365	1.000						
EFF2	-0.254	-0.317	-0.270	-0.322	-0.282	-0.196	-0.325	0.517	1.000					
EFF3	-0.278	-0.309	-0.347	-0.370	-0.278	-0.245	-0.366	0.535	0.681	1.000				
EFF4	-0.294	-0.300	-0.307	-0.396	-0.298	-0.212	-0.380	0.536	0.585	0.625	1.000			
EFF5	-0.248	-0.246	-0.227	-0.275	-0.375	-0.205	-0.306	0.513	0.564	0.519	0.528	1.000		
EFF6	-0.275	-0.256	-0.272	-0.232	-0.212	-0.314	-0.289	0.451	0.437	0.539	0.432	0.479	1.000	
EFF	-0.391	-0.334	-0.351	-0.403	-0.339	-0.294	-0.438	0.900	0.670	0.745	0.807	0.665	0.642	1.000

Note: All correlations are significant at the 0.01 level.

Table C4. VIF Statistics for Formative Indicators

Construct	Indicator	R_i^2	VIF
Software Team Responsive Extensiveness	EXT1	0.543	2.19
	EXT2	0.676	3.09
	EXT3	0.723	3.61
	EXT4	0.651	2.87
	EXT5	0.542	2.18
	EXT6	0.527	2.11
Software Team Responsive Extensiveness	EFF1	0.415	1.71
	EFF2	0.548	2.21
	EFF3	0.591	2.44
	EFF4	0.492	1.97
	EFF5	0.441	1.79
	EFF6	0.361	1.56

Note: The VIF statistic for a formative indicator X_i is calculated by the following formula: VIF (X_i) = 1 / (1 – R_i^2), where R_i^2 is the coefficient of determination of the regression equation $X_i = \alpha_1 X_1 + \alpha_2 X_2 + \alpha_3 X_3 + ... + \alpha_k X_k + e$.

Appendix D

Test of a Second-Order PLS Model

In this second-order model, the *software project performance* construct is modeled as a second-order latent variable consisting of three first-order latent variables (on-time completion, on-budget completion, software functionality). Due to the lack of PLS capability to directly test second-order models, we separately tested the first-order constructs and then used the computed first-order factor scores as manifest indicators

of the second-order construct (Yi and Davis 2003). Since on-time completion and on-budget completion are modeled as single-indicators, only software functionality needs to be tested separately to obtain its factor score.

As shown in Figure D1, both team response extensiveness (.234, $p < .01$) and team response efficiency (.472, $p < .01$) have a significant positive effect on the second-order software project success construct. Team response extensiveness and team response efficiency collectively explain 17.6 percent of the variance in software project performance. Team autonomy and team diversity collectively explain 14.3 percent of the variance in response extensiveness, whereas team autonomy, team diversity, and response extensiveness collectively explain 27.2 percent of the variance in response efficiency. The coefficients for other paths remain stable between the second-order and the first-order models, suggesting the robustness of the research model.

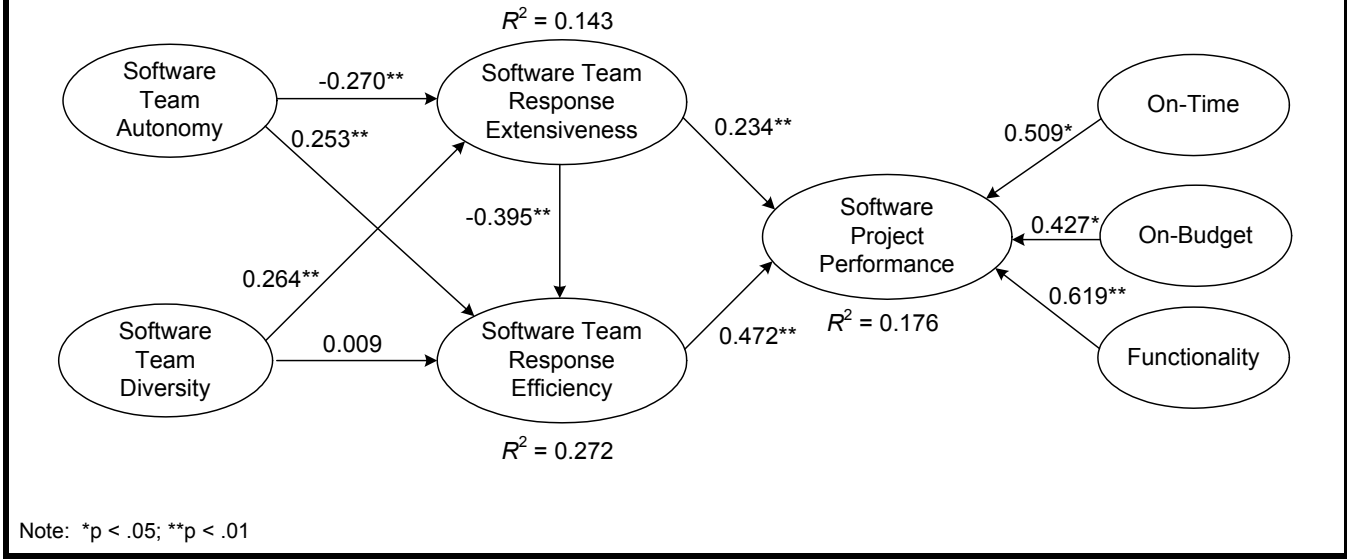

Figure D1. Results of the Second-Order PLS Model

Copyright of MIS Quarterly is the property of MIS Quarterly & The Society for Information Management and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.