Information Systems Research
Vol. 20, No. 3, September 2009, pp. 377–399
ISSN 1047-7047 | EISSN 1526-5536 | 09 | 2003 | 0377

DOI 10.1287/isre.1090.0238

A Control Theory Perspective on Agile Methodology Use and Changing User Requirements

Likoebe M. Maruping, Viswanath Venkatesh
Information Systems Department, Sam M. Walton College of Business, University of Arkansas, Fayetteville, Arkansas 72701 {lmaruping@walton.uark.edu, vvenkatesh@vvenkatesh.us}

Ritu Agarwal
Decision, Operations and Information Technologies, Robert H. Smith School of Business, University of Maryland, College Park, Maryland 20742, ragarwal@rhsmith.umd.edu

In this paper, we draw on control theory to understand the conditions under which the use of agile practices is most effective in improving software project quality. Although agile development methodologies offer the potential of improving software development outcomes, limited research has examined how project managers can structure the software development environment to maximize the benefits of agile methodology use during a project. As a result, project managers have little guidance on how to manage teams who are using agile methodologies. Arguing that the most effective control modes are those that provide teams with autonomy in determining the methods for achieving project objectives, we propose hypotheses related to the interaction between control modes, agile methodology use, and requirements change. We test the model in a field study of 862 software developers in 110 teams. The model explains substantial variance in four objective measures of project quality—bug severity, component complexity, coordinative complexity, and dynamic complexity. Results largely support our hypotheses, highlighting the interplay between project control, agile methodology use, and requirements change. The findings contribute to extant literature by integrating control theory into the growing literature on agile methodology use and by identifying specific contingencies affecting the efficacy of different control modes. We discuss the theoretical and practical implications of our results.

Key words: agile methodologies; agility; control theory; requirements uncertainty; software development; teams
History: Sandra Slaughter, Senior Editor and Associate Editor. This paper was received on June 6, 2007, and was with the authors 9 months for 3 revisions. Published online in *Articles in Advance* August 25, 2009.

Introduction

It is almost a truism that software development is a highly consequential activity for business and society. However, despite over five decades of experience with software development, the process continues to be challenging for development teams. One aspect of the challenge is simply that software development is inherently a complex activity that is embedded with interdependencies, requires the collective input of multiple individuals with often nonoverlapping knowledge sets, and entails significant coordination and project management. A second and perhaps more crucial aspect of the challenge is simply that *what* the software is required to do, i.e., its functionality, is a moving target (Lee and Xia 2005, Nidumolu 1995). Such user requirements changes are largely fueled by continuously evolving business needs (Cusumano and Yoffie 1999, Hoorn et al. 2007, Iansiti and MacCormack 1997) and, in addition to being unpredictable, they are occurring with increasing frequency and speed in an ever more competitive market environment (Iansiti and MacCormack 1997).

An inability to respond to changing user requirements has been implicated as one cause for major project failures, including outcomes such as budget cost overruns, poor product quality, and project schedule overruns (Standish Group 2003). It is no wonder that requirements change is often viewed as a significant threat to software development project success (Boehm 1991, Hoorn et al. 2007, Mathiassen et al. 2007, Nidumolu 1995, Standish Group 2003). To respond to this exigency, the design science and software engineering communities have proposed a set of *flexible* techniques, namely agile methodologies,

that empower software development teams in the face of challenges posed by changing user requirements (Baskerville et al. 2002, Fowler and Highsmith 2001, Highsmith and Cockburn 2001). Fundamentally, these methodologies are purported to imbue flexibility in software development projects, thereby enabling software development teams to perform more effectively.

The importance of flexibility in software development processes—so that they can be more responsive to requirements changes—is underscored in a significant body of research (e.g., Byrd and Turner 2000, Duncan 1995, Gefen and Keil 1998, Lee and Xia 2005, MacCormack et al. 2001). It is evident that in the presence of unpredictability, adaptation is necessary (Conboy 2009). For instance, MacCormack et al. (2001) found a positive relationship between flexible development architecture and software project performance. Likewise, Lee and Xia (2005) reported a positive relationship between team flexibility and end-user satisfaction with a system. However, despite these advances in our understanding of requirements uncertainty and software development project performance, there remain significant gaps in the information systems literature. Although it is accepted that flexibility is desirable in situations where requirements change, flexibility is not without cost, as it is fundamentally inimical to the degree of structure embedded in a development process.

Previous literature has emphasized the importance of structure in software development, offered through development methodologies (Fitzgerald 2000). Indeed, the entire methodology movement in the 1970s and 1980s was predicated on the importance of structure in software development and the need to limit "free-wheeling" by team members (Boehm 1981). A review of the team literature also suggests that autonomy, which affords individual team members the freedom to act of their own volition, may be detrimental for project teams (Cohen and Bailey 1997). However, autonomy has also been identified as an important factor in enabling teams to respond to change (Gerwin and Moffat 1997). It is unclear, therefore, how these seemingly polarized notions should be reconciled in software development teams, where elements of both are necessary for project success. In particular, the extant literature offers limited guidance regarding the governance of agile software development teams,[1] and how project leaders should manage the balance between structure and autonomy.

The purpose of this research is to examine the management of agile software development teams. Specifically, we draw upon control theory to investigate the complex interplay between project management, agile methodology use, and requirements change. Research in the software and information systems development literatures has emphasized the central role played by project management approaches in ensuring the success of development efforts (e.g., Barki and Hartwick 2001, Guinan et al. 1998, Kirsch 1997, Sillince and Mouakket 1997), highlighting the importance of project leaders in influencing software development teams' progress toward achieving desirable project outcomes (Guinan et al. 1998, Kirsch 1997). Control theory is the primary theoretical lens through which the process of guiding teams to project completion has been understood (Henderson and Lee 1992, Kirsch 1997).

This research contributes to the software development literature in several ways. First, by integrating control theory, we add to the growing literature on agile software development and shed light on the managerial mechanisms that can best support the use of such methods in software development teams. Second, although prior research has studied control mechanisms, no work we are aware of has examined control in the context of different environmental and internal team conditions. We extend control theory by identifying contingencies affecting the efficacy of different control modes. We examine the relationships between agile methodology use, control modes, and requirements change in a field study of 862 software developers in 110 teams, using objective measures of project quality. Finally, as noted earlier, although knowledge about how to manage traditional software teams has accumulated over the past few decades, the governance of agile development teams has not received much attention. Indeed, as noted by Boehm and Turner (2005), the incorporation of agile methods into traditional software development environments poses many new challenges, highlighting the

[1] We use the term, "agile software development teams," to refer to teams that are using an agile methodology.

need for a better understanding of people and process issues. This research provides a granular understanding and yields pragmatic guidance for project leaders whose software development teams use agile methods to varying degrees.

Theoretical Background

Control in Software Development Projects

Although the study of the management of software development teams using agile methodologies is in a nascent stage, prior research provides some insights on factors of theoretical and pragmatic importance. A major problem that software project leaders continue to face is how to effectively manage *team* work in the software development process (Barki and Hartwick 2001). Software development has been characterized as not only a technical process, but a social process as well, requiring the effective management of relationships to facilitate the utilization of critical skills and expertise (Beath and Orlikowski 1994). Hence, software project leaders must make important decisions about the appropriate methods for managing both technical and social processes. Control theory, which has tended to focus on the management of individual employees (Ouchi 1979), has provided important insights into understanding the management of software development teams. In the context of software development, *control* is defined as management's "attempts to ensure that individuals working on organizational projects act according to an agreed-upon strategy to achieve desired objectives" (Kirsch 1996, p. 1). Control is generally exercised through the monitoring and evaluation of behaviors or outcomes, and has been identified as an important antecedent of project team performance in terms of both efficiency and effectiveness (Henderson and Lee 1992, Nidumolu and Subramani 2003).

Control modes are generally categorized into two types, formal and informal. *Formal control* modes are those exercised by management through formal documentation (Kirsch 1996). They are viewed as a strategy for evaluating and rewarding performance in an organizational context (Eisenhardt 1985, Kirsch 1997). Through formal control, management is able to set specific standards against which software development team performance will be evaluated, and teams are rewarded based on how well they meet these performance standards. It is generally expected that formally setting performance standards encourages software development teams to align their goals with the outcomes desired by the organization (Kirsch et al. 2002). Two specific forms of formal control are outcome control and behavior control. *Outcome control* involves outlining a set of project goals to be achieved; and rewards are made contingent on the accomplishment of goals (Kirsch 1996, Ouchi 1979). Thus, when formal control is exercised in the form of outcome control, the emphasis is on software development team *outputs* (e.g., project deadlines, defect rates), regardless of the process used to achieve them (Henderson and Lee 1992). In contrast, *behavior control* emphasizes the behaviors, processes, or procedures that software development teams must follow in order to achieve project goals (e.g., standard operating procedures, development methodologies). To the degree that the enactment of prespecified processes is expected to yield desired project outcomes (Kirsch 1997), rewards are, therefore, made contingent on software development teams' adherence to prespecified work processes (Henderson and Lee 1992, Kirsch 1996). The ability to exercise behavior control is predicated on management's ability to perfectly observe and understand the process through which software development teams turn inputs into outputs (Kirsch et al. 2002).

Whereas formal control modes represent a performance evaluation strategy for aligning employee goals with organizational goals, *informal control* modes are viewed as a social or people strategy for regulating employee goals (Jaworski 1988). Rather than relying on formal documentation, informal modes of control emphasize social dynamics and self-regulation as a way of reducing employee-organization goal incongruence. Individuals and social collectives, such as teams, take on the responsibility of ensuring that their work is geared toward achieving organizationally espoused goals. Hence, the monitoring function is largely delegated to the employee rather than management (Eisenhardt 1985, Ouchi 1979); however, management can take steps to encourage such self-monitoring behavior via incentives.

Two forms of informal control are clan control and self control. *Clan control* is exercised by socializing team members into a specific set of norms and values that are valued by the organization. Although management espouses the desired values and norms, members of the social collective reward behavior that is aligned with those values and sanction behavior that is inconsistent with espoused values (Ouchi 1979). Thus through shared rituals and experiences, acceptable behaviors are socially reinforced (Kirsch 1996). For example, if on-time project delivery is valued then team members who work overtime to ensure timely project completion are rewarded for such behavior. The exercise of clan control grants software development teams a degree of autonomy in identifying important project goals and determining how those goals are attained. However, management attempts to influence this social form of self-regulation to ensure goal alignment.

In contrast to clan control, which emphasizes collective regulation of goals and behavior, *self control* is geared toward the individual and represents the extent to which individuals have the autonomy to "determine both what actions are required and how to execute these activities" (Henderson and Lee 1992, p. 760). Self control encourages individuals to set their own goals and then self-regulate and self-monitor their progress in achieving those goals (Henderson and Lee 1992, Manz and Sims 1987). It is a function of the objectives and standards that individual employees set for themselves and operates outside the scope of formal and clan controls (Kirsch 1996, 1997). Although individuals are primarily responsible for exercising self control, management can promote it through the provision of incentives (Kirsch 1996) or through the selection of self-motivated individuals (Ouchi 1979). Indeed, Kirsch et al. (2002) note that self control can be enabled by establishing incentive schemes that reward autonomy and self-regulation. Tasks such as software development that are inherently knowledge-intensive and demand creativity and intellectual activity are particularly well-suited for the exercise of self control (Henderson and Lee 1992).

It is important to note that, although the control modes outlined above have been discussed in isolation, they are not exercised independently and project leaders frequently deploy different combinations of controls in software development projects—creating a portfolio of controls (Choudhury and Sabherwal 2003, Kirsch 1997). Such portfolios typically include both formal and informal mechanisms and involve the combination of primary and secondary controls (Choudhury and Sabherwal 2003, Jaworksi et al. 1993, Kirsch 1997). There is often a greater reliance on formal modes of control, which are supplemented with informal controls (Kirsch 1997). For example, a project leader may specify precise project deadlines to be met (a formal control) and also provide bonuses or special recognition for developers who create and adhere to a strict plan for achieving important project milestones (an informal control). Irrespective of the specific portfolio of controls employed in a project, however, an important characteristic of the various control modes discussed in the literature is the degree to which they afford autonomy to software development teams in managing the software development project—a factor that we argue is critical for agile software development teams.

Agile Software Development Teams

From structured development techniques to rapid application development to object-oriented design, software engineers have continually sought new methodologies and development approaches to address an evolving market for software. Persistent limitations of extant approaches in effectively addressing requirements change have led to the emergence of a set of software development methodologies collectively referred to as agile methodologies (Fowler and Highsmith 2001). Some well-known agile methodologies include eXtreme Programming (XP) (Beck 1999), Scrum (Rising and Janoff 2000, Schwaber and Beedle 2002), Feature Driven Design (Coad et al. 1999), Test Driven Development (Beck 2003), Crystal (Cockburn 2001), and Lean Programming (Poppendeick 2001). Agile methodologies are argued to provide flexibility in software development, thus enabling software development teams to cope with an unpredictable and changing environment (Beck 1999). Although the use of agile methodologies in organizations is still largely experimental and has not yet gained widespread adoption (Fitzgerald et al. 2006), eXtreme Programming is the most widely adopted approach (Baskerville et al. 2002). As with most software

development methodologies, software development teams have tended to use an à la carte approach in their deployment of agile methodologies—implying that they typically adopt a subset of practices rather than all practices of a methodology (Fitzgerald 2000, Fitzgerald et al. 2006).

The key value proposition of agile methodologies is that they emphasize and facilitate flexibility in software development (Baskerville et al. 2002, Beck 2000, Conboy 2009, Fowler and Highsmith 2001), thereby enabling software development teams to effectively respond to any requirements changes that might occur during a project. Although agile methodologies differ in the specific practices they embody, they share common characteristics that facilitate flexibility including an iterative approach to software development, a focus on discrete units of functionality, and an emphasis on simple design (Fowler 2005, Larman 2003). In XP, these characteristics are reflected in practices such as continuous integration and refactoring, whereas Scrum—with its managerial focus—embodies these characteristics in activities such as sprints and daily scrums that a software development team engages in over the course of a project. The practices underlying these agile methodologies recognize that requirements changes are inevitable and thus, attempt to make the process of adapting to changing requirements as efficient as possible. However, the à la carte approach to agile methodology use suggests that software development teams vary in the extent to which they possess the flexibility afforded by such methodologies.

Agile methodologies encourage the delegation of authority to software development team members including discretion over who can change existing code, task scheduling, and the assignment of members to various tasks (Beck 2000). In other words, agile methodologies suggest that team members be largely responsible for managing their own processes, making decisions about how project goals will be attained, and delegating task responsibility to various team members. These characteristics are strikingly similar to the critical characteristics of self-managing teams (Cohen et al. 1996, Hackman 1986). Self-managing teams are those where members "manage themselves, assign jobs, plan and schedule work, make production- or service-related decisions, and take action on problems" (Kirkman and Shapiro 2001, p. 557). Unlike traditional teams, self-managing teams are largely responsible for management and decision making; activities that are typically the purview of management (Manz and Sims 1987). Thus, autonomy is a critical factor in determining how well teams are able to self-manage (Hackman 1986, Langfred 2004). Software development teams using agile methodologies possess many of these characteristics as reflected by teams that were examined by Fitzgerald et al. (2006). These development teams were responsible for management and decision making over the course of the project conducted at Motorola (Fitzgerald et al. 2006).

Although agile software development teams possess many of the qualities of self-managing teams, the conditions under which they operate make them particularly unique. One key characteristic of the software development team environment is that they often face high levels of environmental uncertainty as customer requirements and specifications change over the course of a project, thus, making flexibility important (Aoyama 1998, MacCormack et al. 2001, Nidumolu and Subramani 2003). In addition, agile methodologies involve short cycle times, placing pressure on software development teams to create functional units of software in short iterations (Beck 2000). These conditions make it important for project managers to strike a balance between providing the autonomy needed for enhancing team flexibility and providing the structure needed to guide software development teams to successful project completion.

In summary, we identified formal and informal control modes as a key governance mechanism through which project managers can guide software development teams in their work. We also highlighted the emergence of agile methodologies, which have been constructed to specifically address the challenges posed by shifting and evolving user requirements in the context of software products by enabling flexibility in software development. Although it is broadly recognized that the outcomes of a software development project will be affected by the exercise of control, the use of agile methodologies and the extent to which project requirements change, we seek to understand how these factors interact in their impacts on software development team performance. Next we

develop specific hypotheses related to this interplay between these antecedents of performance.

Hypothesis Development

Agile Methodology Use and Project Quality

Agile methodologies are not solely comprised of routines for responding to requirements changes. Rather, the practices that underlie agile methodologies are designed to produce high quality code—i.e., code with few bugs and low complexity (Bieman 2002, Darcy et al. 2005). For instance, the practice of test-driven development emphasizes software testing over the course of a development project (Beck 2003), thus producing code with relatively fewer bugs, even under conditions in which user requirements remain stable. Likewise, the refactoring practice in XP creates simplified code and reduces errors, and the pair programming practice enables more code to be developed with fewer errors (Beck 2000, Nosek 1998). In sum, agile methodologies comprise practices that serve the dual purpose of producing high quality code and providing the ability to respond to change. Teams that employ the practices outlined in agile methodologies are well positioned to produce high quality code. Thus, we hypothesize:

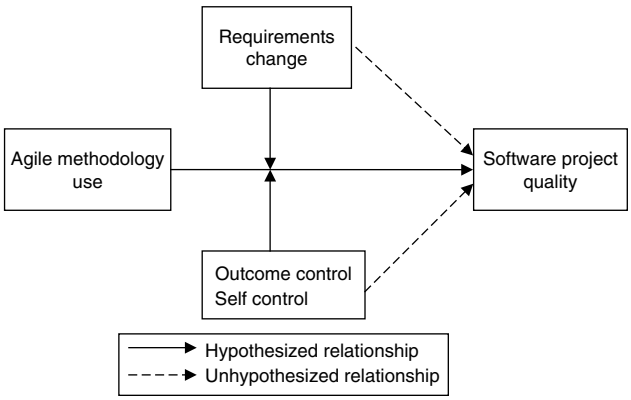
Hypothesis 1. *Agile methodology use will have a positive influence on software project quality.*

User requirements for software specification are typically gathered at the beginning of software development projects, and software development teams strive to get an accurate understanding of customer needs so that they can build software that meets those needs. However, as noted earlier, it is quite common for user requirements to change over the course of a software development project (Walz et al. 1993). Customers often do not have a clear understanding of their needs and identify additional needs as the project progresses, or change their minds about previously stated needs. These changing needs may require software development teams to make changes to the inputs into a system, the outputs from the system, the type of data that the system can process, and the way data are processed (Lee and Xia 2005). Because they are unpredictable, requirements changes take significant time and effort to respond to and can negatively affect the quality of the software. Indeed, previous research has found a negative relationship between requirements changes and project quality (e.g., Curtis et al. 1988, Nidumolu 1995).

The ability to effectively respond to requirements changes is a key factor that differentiates high performing teams from teams that do not perform as well. (Lee and Xia 2005, MacCormack et al. 2001). Agile software development teams anticipate that requirements will inevitably change over the course of a project and have processes in place for effectively managing such changes when they do occur (Beck 1999, 2000). The ability to respond to requirements changes is embedded within the specific processes that make up each agile methodology (Conboy 2009, Fowler and Highsmith 2001, Larman 2003). Flexibility is not expected to be important when user requirements remain fairly stable over the course of a software development project because there is less pressure on the development teams to respond to change. However, as requirements changes increase in their frequency, the flexibility enabled due to the use of agile methodologies becomes a key component for effective response (Beck 1999, MacCormack et al. 2001). Hence, other things being equal, software development teams using agile methodologies should produce software of better quality than that of teams that do not, when requirements changes are high.

As Hypothesis 1 suggests, we expect agile methodology use to have a positive relationship with software project quality. We further expect this relationship to be enhanced in the presence of requirements change. However, to the extent that the exercise of control defines the context within which software development teams operate, the relationship between agile methodology use, requirements change, and software quality is arguably more complex. The exercise of control is expected to moderate the relationship between agile methodology use and software project quality at various levels of requirements change. In other words, the control modes that project leaders employ should play a significant role in determining the degree to which agile methodology use will enable software development teams to cope with change. This is the core logic underlying our research model shown in Figure 1. Next, we propose specific

Figure 1 Conceptual Model of Control Modes, Agile Methodology Use, and Requirements Change



relationships between agile methodology use, control modes, and requirements changes in influencing software project quality.

Control Modes, Agile Methodology Use, and Requirements Changes

Outcome controls provide a well-defined, unambiguous, and supportive context for achieving software development project goals. Project leaders often set performance standards related to project quality, meeting project deadlines, and managing project budgets (Humphrey 1995, Kirsch 1996), and frequently, clients have a hand in determining performance standards (Kirsch et al. 2002). These outcome-focused controls constitute clear benchmarks against which software development teams can monitor their progress. In the complex endeavor that is software development, it is often easier to specify outcomes than to monitor behavior (Kirsch 1996, 1997). Because of their emphasis on output rather than process, such controls give software development teams the autonomy to determine the best way to deploy resources in an effort to meet project goals (Nidumolu and Subramani 2003). This autonomy becomes even more consequential when software development teams need to respond to requirements changes (Gerwin and Moffat 1997, Henderson and Lee 1992).

As noted earlier, agile methodologies provide the necessary mechanisms for team self-regulation. Task delegation and coordination is managed through practices such as collective ownership, coding standards, and pair programming (Beck 2000, Fitzgerald et al. 2006). Through such practices, software development teams can effectively prioritize important tasks and determine the appropriate resources to deploy in managing those tasks. Actual task execution and software quality control are embedded in practices such as continuous integration, refactoring, and unit testing. These practices are designed to ensure software design simplicity and high quality code (Beck 1999, 2000). The decentralized decision-making authority is especially important when requirements are volatile (Nidumolu and Subramani 2003). It ensures that software development teams are able to take effective action on problems as they arise. The exercise of outcome control creates an environment for such autonomy while also providing clear performance goals (Henderson and Lee 1992).

Outcome controls are especially important as requirements change increases because of the need for the development team to respond. Gerwin and Moffat (1997) posited that a lack of autonomy would negatively influence performance in new product development teams, and empirically found a negative correlation between a lack of autonomy and team performance. Outcome control enables software development teams to maintain their focus on achieving objectives; an essential element when teams need to be responsive to requirements changes. Because of the flexibility inherent in agile methodologies, software development teams are likely to consider many alternative approaches to meeting user needs. There is a risk, therefore, that as teams strive to identify alternative solutions in responding to requirements changes, they might lose sight of project objectives. Outcome control, with its emphasis on task outcomes, mitigates this risk and ensures that software development teams maintain their focus on meeting objectives (Kirsch 1997). Established performance criteria provide a baseline against which modified software can be continuously measured (Henderson and Lee 1992). For instance, Kirsch (1997) found outcome control enabled software development teams to maintain their focus on project quality because their performance was contingent on the quality of the system delivered. Without outcome control, agile software development teams may still be able to respond to requirements changes, but without guidance they

may miss key objectives in the process. Thus, we hypothesize:

HYPOTHESIS 2. *The positive relationship between agile methodology use and software project quality is moderated by outcome control and requirements change such that the relationship is more positive when both outcome control and requirements change are high than when either or both are low.*

When managers do not observe the day-to-day activities and behaviors of developers, either because of situational contingencies or by choice, they may encourage the exercise of informal forms of control. Informal controls have been found to be useful in promoting effectiveness in software development projects (Henderson and Lee 1992). In particular, self control, which emphasizes individual self-regulation, has been found to yield positive outcomes in software development (e.g., Bailyn 1985, Henderson and Lee 1992, Weinberg 1971). As is evident, the use of self control as a mechanism provides developers with autonomy and discretion with regard to how tasks are accomplished (Henderson and Lee 1992, Kirsch et al. 2002). For example, Henderson and Lee (1992) suggest that through self control individual developers can reallocate their efforts and choice of methods on tasks without consulting with the project leader. However, the extent to which the exercise of self control is effective when requirements changes are high is not well understood.

As desirable as autonomy is when requirements uncertainty is high, the provision of such autonomy through self control can have detrimental effects in a team setting (Wageman 1995). Significant task interdependencies exist in software development projects, and the effective use of any methodology requires coordinated effort among software development team members (Boehm 1981). This is especially true when requirements changes are high. Under such conditions, team members need to effectively coordinate their efforts in developing solutions while performing continuous integration, refactoring, and pair programming tasks. Because the effectiveness of these practices in responding to change hinges on collaborative effort, an emphasis on self-regulation is potentially inimical to deriving value from agile methodology use. With each team member adopting their own approach to managing interdependent tasks, there is likely to be much disagreement among team members regarding how best to respond to requirements change. For instance, developers charged with programming additional functionalities into software may approach the coding process differently, resulting in incompatible modules and adversely affecting project quality. Hence, although self control ensures that developers self-regulate their behavior with regard to task accomplishment, there may be divergent approaches to the actual accomplishment of tasks. Consequently, left to their devices, the goals of different developers may be highly incongruent, in which case development efforts might not converge to improve software project quality. The use of agile methodologies under such circumstances will prove ineffective. We expect that the use of self control will undermine the benefit of agile methodology use in software development teams when requirements change is high. Thus, we hypothesize:

HYPOTHESIS 3. *The positive relationship between agile methodology use and software project quality is moderated by self control and requirements change such that the relationship is less positive when self control and requirements change are both high than when either or both are low.*

In summary, the research hypotheses are constructed to answer the critical question: under what contingencies is the value of agile methodology use in influencing software project quality enhanced? We argued that requirements change and project governance in the form of control constitute two significant contextual conditions that exhibit important moderating influences on the relationship between agile methodology use and software development team performance. One important assumption underlying our theorizing is that, consistent with prior work that has posited and found empirical evidence for the existence of portfolios of control, we are not suggesting that formal controls and informal controls are substitutes. Rather, we expect both to be simultaneously used to varying degrees by leaders of software development teams. However, we leave explicit theorizing about the relative trade-off between formal and informal control to future work and focus here solely on the independent effects of each in conjunction with requirements change and agile methodology use.

Method

We conducted a field study of software development teams to test the hypotheses. The study spanned a little over three months and included three different points of measurement. The participants, measurement, data collection procedure, and analysis plan are discussed in this section.

Participants

The participants were employees in a major U.S.-based consulting firm. The participating firm has over 20,000 employees and serves a broad-based clientele spanning multiple industries including banking, healthcare, insurance, and retail. Specifically, participants were all members of active software development project teams. The teams that participated in the study were launching software development projects for a major U.S.-based client company. In the study, 151 teams agreed to participate, and a total of 862 employees in 110 software development project teams provided usable responses at all three points of measurement, for an effective response rate of 72.8%. Although it was desirable for all teams to provide responses at all measurement points, the study duration made this practically infeasible. Of the participants, 230(26.7%) were women. The average age of the participants was 29.6 (s.d. = 5.52). On average, the participants had 5.3 years of programming experience (s.d. = 2.25). The average team size in the study was 7.84 (range: 7–10).

Measurement

To the extent possible, we operationalized constructs by adapting existing scales to the context of the current study. A few of the constructs were measured from all team members within each team. Because the level of analysis is the team, it is necessary to ensure that the aggregation of individual-level scores to form the team-level construct is appropriate (Bliese 2000). Therefore, where relevant, we report on the within-group agreement index ($r_{wg(j)}$), and intra class correlation coefficients (ICC). The $r_{wg(j)}$ reflects the extent to which individual item responses in a team converge greater than would be expected by chance (James et al. 1984). The ICC(1) reflects the degree to which there is between-group variance in individual responses (Bliese 2000). The ICC(2) reflects the stability of the group-level means across the sample (Bliese 2000). Finally, one of the key constructs in the model—agile methodology use—did not have an existing measure. It was, therefore, necessary to develop a new scale to measure the construct.

Agile Methodology Use. To our knowledge, there are no existing measures for assessing the use of agile practices in software development teams in the field; therefore, we developed a new scale to operationalize this construct. The six key XP practices used to reflect agile methodology use are pair programming, continuous integration, refactoring, unit testing, collective ownership, and coding standards.[2] Pair programming, continuous integration, refactoring, and unit testing are action-oriented agile practices (Beck 2000). These programming related practices are guided by an agreement about collective ownership of code and the use of coding standards (Beck 1999). We generated item pools to capture the use of each practice. We were careful to ensure that the items reflected the level of analysis for which they were developed—i.e., the team as the referent in each item (Chan 1998). Klein et al. (1994) point out the importance of ensuring alignment between theory and measurement. Content validity of the items was ensured by basing the items on definitions and descriptions of the XP practices (Straub 1989). Developers who had experience with using XP also provided suggestions for refining the items and adding others. After following scale development procedures suggested by DeVellis (2003), we pilot tested the scales in a field sample of 149 developers.[3] The sample consisted of developers who had experience with the XP methodology and were involved in ongoing projects or recently completed software development projects. There was adequate convergent and discriminant validity in the scales.

As noted earlier, responsiveness to change is the raison d'être of agile methodologies. However, as teams

[2] Although we measured the use of all XP practices, we selected these six practices that reflect activities that specifically promote flexibility and have been identified as instrumental in enabling software development teams to respond to requirements changes (Larman 2003).

[3] In the interest of brevity, details of the scale development process are omitted here and are available, upon request, from the first author.

use agile practices to varying degrees, their development processes exhibit different levels of flexibility. To accurately reflect this nuance, we operationalized agile methodology use as an additive index of the six XP practices. As individuals within each team constituted the respondents, it was necessary to aggregate individual responses to compute a team-level score for each team. All six agile practices had a reliability greater than 0.70 and the average $r_{wg(j)}$ value for each scale was above 0.70. The random effects ANOVA-based F-statistic for each scale was significant at $p < 0.001$, indicating statistically significant between-team differences in individual responses to each scale. The ICC(1) for each scale was greater than 0.14, suggesting meaningful between-team variation in the measures. Further, the team-level means for each scale demonstrated adequate stability, with all scales yielding an ICC(2) greater than the recommended 0.70 value (Bliese 2000). This assessment suggested that it was appropriate to compute team-level scores for the six XP practices by aggregating individual within-team responses. As anticipated, the teams varied greatly on the extent to which they used agile methodologies.

Control Modes. Measures for the control modes were adapted from Kirsch (1996). Each scale was measured on a seven-point Likert agreement scale. The scale for outcome control included six items that had a reliability of 0.82. The measures capture the degree to which established standards were used to evaluate project performance. The exercise of self control was measured via a three-item scale that assessed the degree to which developers were rewarded for their individual performance on project tasks. The scale had a reliability of 0.85. As expected, there was variation in the degree to which these different control modes were employed across the software development teams in the sample.

Requirements Change. We used Nidumolu and Subramani's (2003) three-item scale to measure requirements change. The scale captures the extent to which user requirements changed over the course of a software development project from start to finish. The scale had a reliability of 0.79. Consistent with Nidumolu and Subramani (2003), the scale was measured on a seven-point Likert agreement scale. The mean value for requirements change was 5.01 with a standard deviation of 1.33. This indicates that there was substantial variation in requirements change across software development teams in the sample.

Project Quality. We used bug severity and software complexity as objective measures of project quality. Archival project data on the number of bugs and the number of hours required to fix them were obtained. We computed bug severity as the product of these values: lower bug severity indicates higher software quality. Software complexity is an appropriate quality measure as it has long-term implications for customer value (Banker et al. 1998, Card and Glass 1990). A significant portion of a project's cost is in the maintenance, which is tied to software complexity (Banker et al. 1991, Card 1992). Commenting on software quality, Brooks (1987) notes that the best software designers produce code that is simple and clear. High software complexity, therefore, indicates low project quality and vice versa. Consistent with Banker et al. (1998), software complexity was measured in three ways: component complexity, coordinative complexity, and dynamic complexity. *Component complexity* reflects "the number of distinct information cues that must be processed in the performance of a task" (Banker et al. 1998, p. 435) and is computed as the number of data elements that are referenced in the software. *Coordinative complexity* represents the interdependent relationships between information cues in the software and is measured using McCabe's (1976) cyclomatic complexity metric. Finally, *dynamic complexity* reflects changes in cue interdependencies at runtime and is measured as the number of decision paths that are altered at runtime. All three forms of complexity were normalized by dividing them by the number of lines of code (Banker et al. 1998).[4]

Control Variables. We controlled for additional factors that might account for variation in project quality. Prior research suggests that programming

[4] Software complexity can also vary as a function of the specific programming language used. This made it necessary to determine the programming language that was used in each project in our sample. All software development teams used the same programming language—Java. Consequently, we were comfortable that none of the variability in the software complexity measures could be attributed to programming language.

experience is an important predictor of project quality (Banker et al. 1998, Vessey 1989). Therefore, we computed the average number of years of *programming experience* for each team. *Team size* has also been found to affect project quality. Increasing team size can create coordination problems, affecting the quality of team output. Team size was, thus, included as an important control. We also controlled for two key project characteristics: project size and project modularity. *Project size* was measured as the number of lines of code in the final project (Banker et al. 1998). We measured *project modularity* as a count of the number of modules in the software.

Procedure

Data were collected near the beginning, during the middle, and at the end of the software projects in naturally occurring conditions within the organization. The organization launched a new enterprise-wide project involving 151 software development project teams. Each software development team was responsible for developing a specific module for an enterprise-wide system to replace a client organization's existing legacy systems. The development of each module, thus, represented a subproject. The software solutions were expected to provide greater business process support while enabling enhanced cross-unit interoperability in the client organization. Through interviews with the project coordinator and subproject managers, we ascertained that there was no interdependence across the modules (i.e., no two subprojects were dependent upon each other for successful completion) and the organization wanted the teams to function autonomously as it was critical to ensure completion of the modules within the deadline.[5] To triangulate this information, we also conducted interviews with top management in the consulting firm as well as top management in the client firm. All interviewees confirmed that there was no interdependence across subprojects. The software subprojects spanned different application domains. For instance, one subproject involved developing a system to support the client's billing process. Another subproject required a software solution to support the client's order management process. The project was expected to last three months.

All project teams had a project manager who was responsible for managing progress toward project completion. Weekly meetings were held between project managers and their respective teams to discuss project-related issues. Project managers evaluated the output of the team as a whole and also evaluated individual developer performance. Prior literature has consistently pointed out that software development teams frequently adapt and appropriate methodologies in different ways (e.g., Fitzgerald 2000, Fitzgerald et al. 2006). Indeed, arguably methodologies—such as agile development—that are specifically constructed to enable flexibility give software developers more degrees of freedom in deciding how they want to conduct specific software development activities. In our empirical context, although the overall development methodology prescribed for projects was agile development, given that the project teams were autonomous by design, project managers were given discretion over how their team appropriated the methodology. We were aware of this distinction and purposively chose this research site as it offered the desired variability in the use of agile practices. In addition to having discretion over the extent to which their teams used agile methodology practices, each project manager also had discretion over how control was exercised in his or her team.

All project teams began working simultaneously. Participants were informed that the purpose of the study was to examine software development team processes. To organize team responses, all team members and project managers were instructed to agree on a single team name, which would be used to track the questionnaires. The selection of team names was standard practice within the organization. Further, unique bar codes were placed on each survey so that responses could be tracked over time, while maintaining anonymity. One week after the start of the projects, participants filled out the first questionnaire. During this wave of data collection, team members provided demographic information. Project managers responded to questions regarding the use of outcome

[5] In making the subproject assignments, the organization was careful to ensure that each module could be developed as a standalone product without needing input from other modules during development.

Table 1 Correlations and Descriptive Statistics

Variables	Mean	SD	1	2	3	4	5	6	7	8	9	10	11
1. Outcome control	4.58	1.13											
2. Self control	4.22	1.20	−0.12										
3. Agile methodology use	4.98	1.42	0.15*	0.20***									
4. Requirements change	5.01	1.33	−0.17*	0.14*	0.14*								
5. Team size	7.84	1.15	−0.18**	−0.07	−0.18*	0.05							
6. Programming experience	5.30	2.25	−0.04	−0.13*	−0.22**	0.15*	0.24***						
7. Project size	488,344	61,255	0.17*	0.24***	0.15*	0.23***	0.34***	0.22***					
8. Modularity	4.44	1.30	0.20***	0.24***	0.21***	0.15*	0.15*	0.21***	0.19**				
9. Bug severity	28.80	12.77	−0.19**	0.27***	−0.16*	0.18*	0.11	0.04	0.02	−0.31***			
10. Component complexity	0.680	0.112	−0.24**	−0.19**	−0.22***	0.24***	0.19**	0.14*	0.29***	−0.29***	0.28***		
11. Coordinative complexity	0.091	0.022	−0.19**	0.21***	−0.19**	0.24***	0.20***	0.13*	0.24***	−0.30***	0.21**	0.35***	
12. Dynamic complexity	0.017	0.015	−0.21**	0.15*	−0.23***	0.21***	0.22***	0.09	0.30***	−0.29***	0.15**	0.24***	0.29***

Note. $n = 110$.
$^{*}p < 0.05$; $^{**}p < 0.01$; $^{***}p < 0.001$.

and self control to guide team work.[6] Although the use of controls was measured during the early stages of the project, subsequent interviews confirmed that project leaders did not change their choice of control during later stages of the project. Five weeks later, when the teams were well into their software development projects, the second questionnaire was administered. During this second wave of data collection, team members responded to questions about their team's use of XP practices during the project.[7] The third and final questionnaire was administered eleven weeks later. During this final wave of data collection, project managers responded to questions about the extent to which requirements were stable over the course of the software project. Objective data on the software projects were also collected after the projects were completed but shortly before the final product was delivered to the client.[8]

Analysis

The convergent and discriminant validity of variables in the model was assessed using factor analysis with direct oblimin rotation. All items loaded on the prespecified constructs with loadings greater than 0.65 and cross-loadings less than 0.30.[9] The Pearson product moment correlations, means, and standard deviations of the constructs in the model are presented in Table 1. It is interesting to note that agile methodology use is negatively correlated with the four measures of software project quality. Outcome control is also negatively correlated with the four measures of software project quality. In contrast, self control is positively correlated with bug severity, coordinative complexity, and dynamic complexity but negatively correlated with component complexity. Appendix B presents a break down of the descriptive statistics by subgroup (e.g., by agile methodology use, by control mode). The descriptive statistics were split into lower and upper quartiles. We conducted other types of sample splits and found the overall pattern of means to be similar.

In order to test the main effect and moderating hypotheses, we use a seemingly unrelated regression equations approach (Greene 1997, Zellner 1962). We chose this approach because previous research has found a relationship between software complexity and bug severity (Kemerer 1995). This raises the possibility of correlated error terms among the models predicting bug severity and software complexity resulting in biased estimates in ordinary least squares regression. In addition, this approach helps allay concerns about the projects in the sample being part of

[6] To avoid hindsight bias, we decided to assess the exercise of control toward the beginning of the project.

[7] We measured the use of XP practices in the middle of the projects to get an accurate assessment of the practices in use in the software development projects. We wanted to avoid any hindsight bias that might occur if we asked respondents to reflect on their use of XP practices after the projects had been completed.

[8] Requirements change was measured at the end of the project to more accurately capture the extent to which these changes occurred over the course of the entire project.

[9] Given the clean pattern of results and in the interest of space, we have not shown the results here.

a larger project. Consistent with guidelines outlined by Baron and Kenny (1986), we used a hierarchical approach to testing the hypotheses in the model. Control variables are entered into the first block of the regression model. Main effect variables are entered into the second block, followed by the interaction terms in the third block. We scale centered the control mode, agile methodology use, and requirements change variables to reduce multicollinearity between the main effect variables and the interaction terms (Aiken and West 1991).

Results

The results of the moderated regression analysis predicting software project quality are presented in Table 2. Model 1(a, b, c, d) presents the regression equations with only the control variables. The results of the main effects only models are presented in model 2(a, b, c, d). The main effects only models explain 17%, 28%, 25%, and 27% of the variance in bug severity, component complexity, coordinative complexity, and dynamic complexity respectively. Requirements change has a positive effect on bug severity ($\beta = 0.14$, $p < 0.05$), coordinative complexity ($\beta = 0.16$, $p < 0.05$), and dynamic complexity ($\beta = 0.16$, $p < 0.05$). These findings are consistent with prior research indicating that requirements change negatively influences software project quality. However, contrary to expectations, requirements change has a negative effect on component complexity ($\beta = -0.16$, $p < 0.05$). The control modes have significant effects on the four measures of project quality. In particular, outcome control has a significant negative influence on bug severity ($\beta = -0.12$, $p < 0.05$), component complexity ($\beta = -0.17$, $p < 0.01$), coordinative complexity ($\beta = -0.15$, $p < 0.05$), and dynamic complexity ($\beta = -0.21$, $p < 0.01$). In contrast, self control has a positive effect on bug severity ($\beta = 0.14$, $p < 0.05$), coordinative complexity ($\beta = 0.14$, $p < 0.05$), and dynamic complexity ($\beta = 0.13$, $p < 0.05$), and has a negative effect on component complexity ($\beta = -0.13$, $p < 0.05$).

In Hypothesis 1, we predicted that agile methodology use would be positively related with software project quality. As indicated in the main effects models, agile methodology use had a positive influence on software project quality. Specifically, the coefficient for agile methodology use is negative in the regression models predicting bug severity ($\beta = -0.19$, $p < 0.01$), component complexity ($\beta = -0.26$, $p < 0.001$), coordinative complexity ($\beta = -0.22$, $p < 0.001$), and dynamic complexity ($\beta = -0.24$, $p < 0.001$).

The interaction effects model results are presented in model 3(a, b, c, d). The interaction effects models explain 41%, 43%, 43%, and 44% of the variance in bug severity ($\Delta R^2 = 0.24$, $F = 4.16$, $p < 0.001$), component complexity ($\Delta R^2 = 0.15$, $F = 3.06$, $p < 0.001$), coordinative complexity ($\Delta R^2 = 0.18$, $F = 3.67$, $p < 0.001$), and dynamic complexity ($\Delta R^2 = 0.17$, $F = 4.08$, $p < 0.001$) respectively. As the ΔR^2 values indicate, the interaction models explained statistically significantly more variance in the predictors over and above that explained by the main effects model, thus, providing support for Hypotheses 2 and 3 (Carte and Russell 2003). Further, a power analysis suggested that our sample size of 110 teams was enough to detect medium sized effects with a power of 0.80 and α of 0.05. We examined the variance inflation factors (VIFs) to assess multicollinearity. As indicated in Table 2, the VIFs were all well below the recommended cutoff value of 10 (Ryan 1997), thus suggesting that multicollinearity was not a concern in the results. The three-way interaction between outcome control, agile methodology use, and requirements change is significant in predicting bug severity ($\beta = 0.30$, $p < 0.01$), component complexity ($\beta = 0.32$, $p < 0.001$), coordinative complexity ($\beta = 0.34$, $p < 0.001$), and dynamic complexity ($\beta = 0.32$, $p < 0.001$). Further, the three-way interaction between self control, agile methodology use, and requirements change has a significant influence on bug severity ($\beta = 0.34$, $p < 0.001$), component complexity ($\beta = 0.35$, $p < 0.001$), coordinative complexity ($\beta = 0.35$, $p < 0.001$), and dynamic complexity ($\beta = 0.37$, $p < 0.001$).

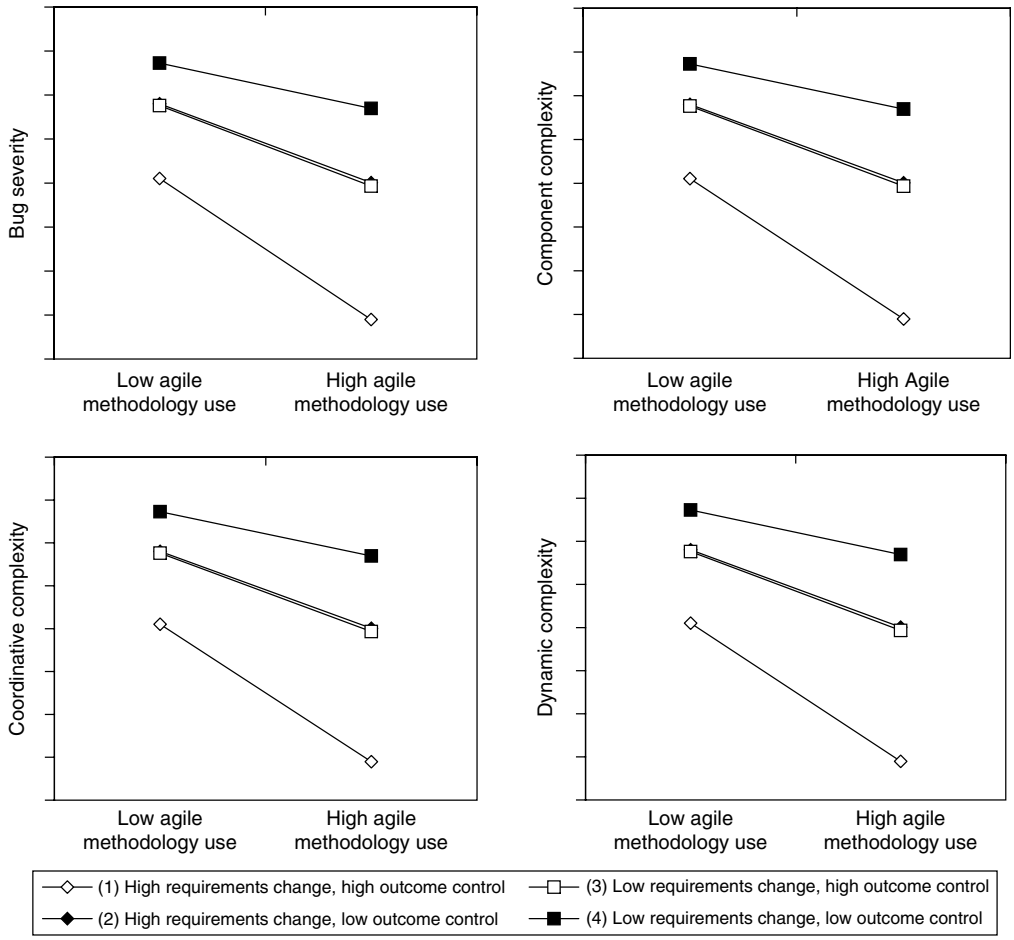
To better understand the pattern of the three-way moderation, we plotted the interactions following guidelines by Aiken and West (1991). Interactions were plotted at one standard deviation above and below the mean for requirements change and each control mode. The interactions are displayed in Figures 2 and 3. As Figure 2 shows, agile methodology use generally has a negative effect on bug severity and software complexity (i.e., a positive influence on project quality). Further, as indicated in Table 2, outcome control has a negative main effect on bug severity and software complexity. This suggests that agile

Table 2 Results of Seemingly Unrelated Regression Equations Models Predicting Bug Severity and Software Complexity

	Bug severity				Component complexity				Coordinative complexity				Dynamic complexity			
Variables	1a	2a	3a		1b	2b	3b		1c	2c	3c		1d	2d	3d	
Controls:																
Team size	0.03	0.01	0.01	(1.40)	0.15*	0.09	0.06	(1.29)	0.15*	0.04	0.02	(1.30)	0.14*	0.07	0.01	(1.44)
Prog. experience	0.03	0.02	0.04	(1.19)	0.08	0.07	0.04	(1.25)	0.04	0.01	0.03	(1.31)	0.04	0.06	0.05	(1.39)
Project size	0.04	0.03	0.02	(1.23)	0.12*	0.05	0.04	(1.20)	0.10	0.02	0.03	(1.25)	0.19**	0.19**	0.12*	(1.35)
Modularity	−0.17**	−0.13*	−0.05	(1.24)	0.17**	0.12*	0.01	(1.30)	0.16*	0.03	0.07	(1.20)	0.07	0.05	0.03	(1.29)
Main effects:																
Agile methodology use		−0.19**	−0.09	(2.11)		−0.26***	−0.17**	(2.01)		−0.22***	−0.19**	(2.22)		−0.24***	−0.13*	(2.13)
Requirements change		0.14*	0.12*	(1.85)		−0.16*	−0.07	(1.34)		0.16*	0.14*	(2.55)		0.16*	0.12*	(1.40)
Outcome control (OCONT)		−0.12*	−0.05	(1.53)		−0.17**	−0.10	(1.95)		−0.15*	−0.12*	(2.90)		−0.21**	−0.18*	(1.57)
Self control (SCONT)		0.14*	0.07	(1.22)		−0.13*	−0.07	(1.38)		0.14*	0.08	(3.09)		0.13*	0.09	(2.85)
Interaction effects:																
Agile methodology use × Req. change			0.06	(2.21)			0.02	(2.59)			0.09	(2.88)			0.01	(1.28)
OCONT × Agile methodology use			0.09	(2.40)			0.04	(1.81)			0.08	(2.35)			0.02	(1.54)
OCONT × Req. change			0.07	(2.31)			0.07	(2.50)			0.10	(3.22)			0.06	(3.20)
OCONT × Agile methodology use × Req. change			0.30**	(2.15)			0.32***	(2.88)			0.34***	(2.44)			0.32***	(2.30)
SCONT × Agile methodology use			0.10	(3.12)			0.08	(3.03)			0.06	(2.10)			0.08	(2.95)
SCONT × Req. change			0.10	(2.80)			0.05	(2.09)			0.10	(1.89)			0.11	(3.09)
SCONT × Agile methodology use × Req. change			0.34***	(1.99)			0.35***	(1.50)			0.35***	(1.95)			0.37***	(2.25)
R^2	0.03	0.17	0.41		0.09	0.28	0.43		0.06	0.25	0.43		0.05	0.27	0.44	
F	4.51*	6.13***	12.58***		3.91*	7.91***	13.75***		4.70*	9.43***	17.60***		3.82*	9.29***	14.24***	
Adjusted R^2	0.00	0.10	0.32		0.05	0.22	0.34		0.02	0.19	0.34		0.01	0.21	0.37	
ΔR^2	0.03	0.14	0.24		0.09	0.19	0.15		0.06	0.19	0.18		0.05	0.22	0.17	
ΔF		2.04***	4.16***			3.20***	3.06***			3.07***	3.67***			3.65***	4.08***	

Notes. $n = 110$. Variance inflation factors are shown in parentheses.
*$p < 0.05$; **$p < 0.01$; ***$p < 0.001$.

Figure 2 Three-Way Interactions: Outcome Control × Agile Methodology Use × Requirements Change

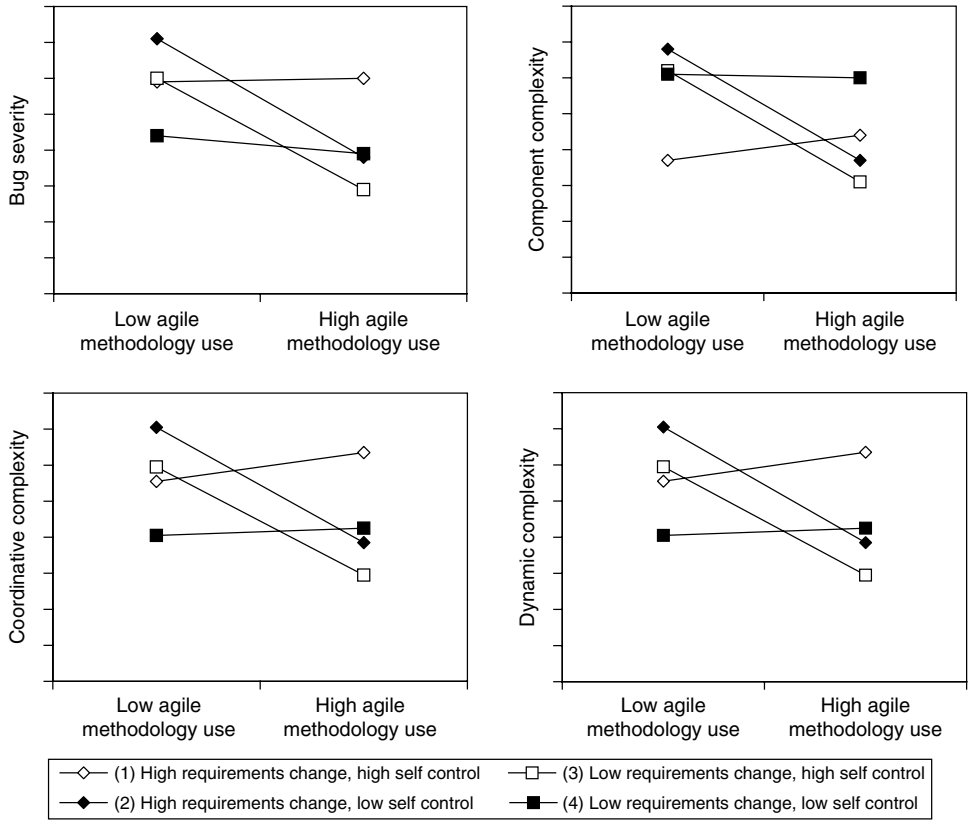


methodology use and the exercise of outcome control are mutually reinforcing of higher project quality. This relationship is borne out in Figure 2. Specifically, outcome control facilitates a stronger negative relationship between agile methodology use and bug severity and software complexity. The joint effect of agile methodology use and outcome control yields the lowest bug severity and software complexity, especially when requirements change is high. Thus, consistent with Hypothesis 2, agile methodology use is most important in improving project quality when outcome control and requirements change are high.

Figure 3 presents the three-way interaction between self control, agile methodology use, and requirements change. As the figure illustrates, agile methodology use is effective in reducing bug severity and software complexity when the level of requirements change is low and self control is high, or when requirements change is high and self control is low. In contrast, agile methodology use is ineffective in reducing bug severity and software complexity when requirements change is high and self control is high. This suggests that the exercise of self control is detrimental to agile methodology use under dynamic environmental conditions. In support of this interpretation, self control generally has a positive main effect on bug severity and software complexity, suggesting that it should have a dampening effect on the influence of agile methodology use under certain conditions. Consistent with Hypothesis 3, the relationship between agile methodology use and project quality is least positive when self control is high and requirements change is high.

As a robustness check, we estimated a few plausible alternative models. Given the order in which the variables in the study were collected, it is possible that the

Figure 3 Three-Way Interactions: Self Control × Agile Methodology Use × Requirements Change



choice of control mode could influence the use of XP, which in turn could influence software project quality. To test this alternative model, we conducted a mediation test following Baron and Kenny (1986). Control modes did not significantly influence the use of XP in our analysis. It is also possible that the use of XP could facilitate more requirements changes, which in turn could affect software quality. Mediational analysis did not provide support for this alternative model. Hence, we are reasonably confident in the robustness of the results.

Discussion

The objective of this research was to theorize and study the complex relationship between control, agile methodology use, and requirements change, and their impact on software development project quality. We sought to understand the contingencies affecting the effectiveness of agile methodology use in software development teams by incorporating considerations of the external environment—via requirements change—and project governance—via control modes. We tested our hypotheses in a field study of software development teams. Our model explained 41%, 43%, 43%, and 44% of the variance in four indicators of project quality: bug severity, component complexity, coordinative complexity, and dynamic complexity respectively. As predicted, agile methodology use had a significant positive influence on project quality and there was a significant three-way interaction between control, agile methodology use, and requirements change in predicting project quality. Each control mode differed in the extent to which it supported or hindered the effectiveness of agile methodology use in the face of requirements change. Taken together, the findings provide insights into the management of software development teams and present a step forward in understanding the tension between the need for structure and the need for autonomy in software development.

Our findings contribute to the literature in several ways. First, we extend the growing literature on

responsiveness to change in software development teams. We noted that although previous research has highlighted the importance of flexibility in enabling software development teams to cope with requirements change (Lee and Xia 2005, MacCormack et al. 2001), less has been said about how such teams should be managed. In this research, we theorized how control modes affect software development project quality by providing a context that either supports or hinders agile software development teams in coping with requirements change. We reasoned that, although autonomy is an important factor in enabling agile software development teams to thrive, the manner in which project leaders facilitate autonomy has implications for how effective the teams can be in completing their assigned task. Specifically, we found that autonomy is most supportive when (1) it is granted to the team as whole, as opposed to individuals within the team; and (2) it is provided in conjunction with specific performance targets for the team.

Second, this research contributes to the control literature. Although previous research has identified key antecedents (e.g., Kirsch 1996, Kirsch et al. 2002) and outcomes (e.g., Henderson and Lee 1992, Nidumolu and Subramani 2003) of various control modes, limited work identifies and theorizes the contingent role that they play in guiding teams toward project completion. We argued that the effectiveness of agile methodology use is contingent, not only on the level of uncertainty wrought by requirements change, but also on the type of control that is exercised. Specifically, we found that agile methodology use is most effective for responding to requirements change when outcome control is exercised. In the case of self control, our results suggest that agile methodology use is least effective in predicting project quality—especially for teams facing high requirements change. Thus, it appears that agile methods are more effective when paired with the exercise of outcome, rather than self control. We reasoned that the exercise of self control that emphasizes self-regulation would create coordination problems in effectively executing agile methodology practices—particularly when the level of requirements change is high. With high levels of requirements change, carefully orchestrated actions need to be executed in a timely manner. This requires effective collaboration on the part of team members, something that is difficult to achieve when rewards are contingent on self-regulation and individual performance.

Strengths and Limitations

Our research has a few key strengths and limitations that need to be highlighted. First, we used a field sample of software development teams, thus, increasing the external validity of the findings. However, the study was conducted within the context of a single organization. Although this allowed us to naturally control for contextual factors that might arise from interfirm differences, future replications across multiple organizations are needed to validate the findings. Second, our study design enabled us to minimize concerns about common method variance (Podsakoff et al. 2003). Specifically, a research design that involved three different points of measurement was used. Independent variables in the research model were measured using survey methods and dependent variables were captured via objective software project metrics. Such a design alleviates concerns about common method variance (Podsakoff et al. 2003). An additional strength of the study is that multiple respondents were used to gather data. Whereas project managers provided responses about control modes, team members responded to questions about their team's agile methodology use. The use of multiple respondents further increases the accuracy of the measures and is instrumental in reducing concerns of common method variance (Podsakoff et al. 2003). In the empirical analysis we operationalized agile methodology use using one specific methodology: XP. Although our theoretical arguments are not based on any one specific methodology, the generalizability of the findings to other agile approaches would need to be empirically verified.

Theoretical Implications and Directions for Future Research

A recent multiple case study by Slaughter et al. (2006) highlights how firms align their software processes, products, and strategies. Slaughter et al. (2006) call for future research to examine the consequences of alignment (or misalignment) at a micro level. This research provides a response to that call. Our findings on the interplay of control modes, agile methodology

use, and requirements change suggest the need for alignment between management strategy and team functioning. Whereas agile methodology use enables software development teams to cope with requirements change, there needs to be a supportive context for meeting such objectives. The modes of control employed by project managers have emerged as an important resource for shaping such a context. Thus, future research needs to incorporate control considerations when examining flexibility in software development teams. In the current research, we examined flexibility as enabled by the use of agile software development methodologies. However, there may be additional ways of enhancing software development team flexibility, including resource fungibility and team design.

Exercise of self control was found to be detrimental to the relationship between agile methodology use and project quality. The use of self control undermined the benefits of agile methodology use in software development teams. This presents an interesting dilemma for managers because software developers are typically compensated for their unique skills and abilities (Ang and Slaughter 2001). Clearly, reward structures that emphasize individual achievement represent an incentive misalignment from a team perspective. This is consistent with Wageman's (1995) finding that teams' whose reward structures are aligned with the level of task interdependence performed better than teams that had incentive misalignments. In software development projects, where task complexity and interdependence are high, outcome interdependence is critical for fostering cooperative behavior. As the results of this study show, when incentives do not promote collective action and coordination, it is difficult to realize the benefits of a flexible development process. Future research should investigate contingencies under which the use of self control yields positive outcomes for project quality.

We studied the interplay between requirements change, control modes, and agile methodology use in the context of XP. Future research would add value by examining how the control modes studied here affect software development teams employing other agile methodologies such as Scrum or Crystal. Agile methodologies differ in the manner through which they provide flexibility, thus, making it important to understand how project management approaches should be tailored to each. Additionally, future research needs to examine these different methodologies at the level of individual practices. It is important to understand the extent to which individual agile practices affect project outcomes, and the contingencies affecting those relationships. Given their importance for supporting interpersonal processes (Maruping and Agarwal 2004), such research should also incorporate the role of communication technologies in supporting the execution of individual agile practices, as an increasing proportion of software development is being conducted by distributed teams.

We did not incorporate temporal considerations into our theorizing. Temporal considerations are another important contingency that should be considered in future research. Recent research on control in software development teams suggests that the control modes employed by project managers can change over the course of a project (Choudhury and Sabherwal 2003, Kirsch 2004). It is possible that as the choice of control mode evolves so too does the effectiveness of agile methodologies in software development teams. Alternatively, changes in control mode might be driven by whether, and how frequently, user requirements change over the course of a software development project. The occurrence of user requirements changes is difficult, if not impossible, to predict ex ante. Therefore, project managers need to make adjustments along the way. Research that sheds light on the interplay between control modes, agile methodology use, and performance over time would make a valuable contribution to the literature. The evolution of control mode use over time may be part of a continuous cycle of alignment as software development teams and project managers strive to improve project outcomes over the course of a project.

Finally, in this research, we only focused on software quality as the outcome measure of interest. However, project performance encompasses a variety of dimensions including client satisfaction and managing costs. It would be important to understand whether and how agile methodology use—at the level of individual practices—enables software development teams to optimize their performance on all aspects of project performance and what control

modes enable them to do so effectively. It is possible that the agile methodology and control modes examined here are more effective for some performance metrics than for others. Future research should examine the impact of these on a broader set of performance metrics.

Practical Implications

A key implication for project managers is the need to match their management approach to the contextual needs of the project. In other words, project managers are advised to strive to achieve alignment between the modes of control they employ and conditions faced by software development teams. If software development teams are able to freeze requirements after the requirements definition phase of a software project, then there may be little need for the flexibility enabled by agile methodologies. Project managers are also better off in using control modes that provide software development teams with autonomy in determining the appropriate methods for managing the software development process, especially when numerous adjustments must be made over the course of a project. This implication raises some interesting questions related to the appropriate skill set of project managers. In essence, our findings point to the need for agility in project management approaches, suggesting that managers need to be well-versed in traditional and more agile methods of development and control.

It is paramount for software development teams to maintain a focus on outcomes throughout a software project. Teams that are continuously working to adapt to evolving user needs can easily lose sight of project objectives. Control modes that emphasize outcomes will help to ensure that software development teams are constantly mindful of project objectives. Project managers can implement such controls through status meetings and reports that track the team's progress toward achieving important project milestones. This type of approach will ensure that all efforts to respond to changing user needs remain in line with project objectives.

Finally, simply providing software development teams with autonomy may not yield desirable results. Managers need to be deliberate about how they provide autonomy through control. When developers are given carte blanche with regard to task accomplishment, the results can be disastrous. Incentive mechanisms that emphasize individual achievement may prompt developers to engage in one-upmanship as they try to showcase their talents. Such behavior may not be in the team's best interests as the solutions that individual developers create may be unduly complex and can prove to be difficult to integrate into existing production code. Managers are therefore encouraged to emphasize team outcome interdependence, rather than individual achievement, in software development project work. An emphasis on collaborative achievement will ensure that efforts to address changing user requirements are in the interest of enabling the team to achieve its objectives.

Conclusions

We developed a model of the interplay between control, agile methodology use, and requirements change, and their effects on software development project quality. Control was argued to be an important contingency affecting the ability of software teams to respond to changing user requirements. We argued that, under conditions of high requirements change, agile methodology use would be important and control modes that provide team autonomy in development activity would be most effective in promoting increased project quality. The hypotheses were empirically tested in a field study. This research is among the first to empirically examine the interplay between control, agile methodology use, and requirements change. The findings offer insights on how to manage software development teams that use agile methodologies.

Acknowledgments

The authors thank the participating organization, which has chosen to remain anonymous, for the generous amount of support provided for this project. The authors gratefully acknowledge the constructive feedback provided by the special issue senior editors and the reviewers. The significant input of Kate Stewart, Paul Tesluk, and Paul Hanges is also gratefully acknowledged. Finally, this paper also benefited tremendously from input by Fred Davis, participants at the ISR special issue workshop in Limerick, Ireland, participants at the brownbag research series at the University of Arkansas, and participants at the research workshop at the Hong Kong Polytechnic University.

Appendix A. Scales[10]

Outcome Control

1. The performance of the team will be evaluated by the extent to which project goals have been accomplished, regardless of how the goals were accomplished.
2. Project goals were outlined at the beginning of the project.
3. Significant weight will be placed upon timely project completion.
4. Significant weight will be placed upon project quality.
5. Significant weight will be placed upon project completion to meet client requirements.
6. Preestablished targets are used as benchmarks for the team's performance evaluations.

Self Control

1. Tangible rewards given to the team, are (or will be) dependent on whether individuals on the team work on their own, without much direction from others.
2. Individuals on this team are rewarded for their individual performance.
3. Individual task performance is rewarded on this team.

Agile Methodology Use

Pair programming

1. How often is pair programming used on this team?[a]
2. On this team, we do our software development using pairs of developers.
3. To what extent is programming carried out by pairs of developers on this team?[a]

Continuous integration

1. Members of this team integrate newly coded units of software with existing code.
2. We combine new code with existing code on a continual basis.
3. Our team does not take time to combine various units of code as they are developed.

Refactoring

1. Where necessary, members of this team try to simplify existing code without changing its functionality.
2. We periodically identify and eliminate redundancies in the software code.
3. We periodically simplify existing code.

Unit testing

1. We run unit tests on newly coded modules until they run flawlessly.
2. Members of this team actively engage in unit testing.
3. To what extent are unit tests run by this team?[a]

Collective ownership

1. Anyone on this team can change existing code at any time.
2. If anyone wants to change a piece of code, they need the permission of the individual(s) that coded it.
3. Members of this team feel comfortable changing any part of the existing code at any time.

Coding standards

1. We have a set of agreed upon coding standards in this team.
2. Members of this team have a shared understanding of how code is to be written.
3. Everyone on this team uses their own standards for coding.

Requirements Change

1. Requirements fluctuated quite a bit in early phases of this project.
2. Requirements fluctuated quite a bit in later phases of this project.
3. Requirements identified at the beginning of the project were quite different from those toward the end.

[10] All items are measured on a seven-point Likert agreement scale, with "Strongly disagree" to "Strongly agree" as anchors unless otherwise noted, with [a] representing the exception that indicates that an item is measured using "Never" to "All the time" as anchors.

Appendix B. Descriptive Statistics by Subgroups[11]

Table B1 Descriptive Statistics by Agile Methodology Use

	High agile methodology use ($n = 28$)		Low agile methodology use ($n = 28$)	
Variables	Mean	SD	Mean	SD
1. Requirements change	5.03	1.30	5.01	1.28
2. Team size	7.79	1.14	7.86	1.18
3. Programming experience	5.22	2.21	5.31	2.27
4. Project size	477,728	60,501	478,845	60,920
5. Modularity	4.40	1.31	4.45	1.28
6. Outcome control	4.55	1.10	4.60	1.16
7. Self control	4.21	1.23	4.28	1.18
8. Bug severity	24.21	10.45	31.22	11.02
9. Component complexity	0.560	0.110	0.710	0.080
10. Coordinative complexity	0.080	0.020	0.113	0.041
11. Dynamic complexity	0.010	0.014	0.022	0.002

[11] Displayed descriptive statistics are based on upper and lower quartiles in the sample of project teams. Other types of sample splits were conducted (e.g., median split, upper third versus lower third split) by each subgrouping. A similar pattern of means resulted for each.

Table B2 Descriptive Statistics by Control Mode

	Outcome control ($n = 28$)		Self control ($n = 28$)	
Variables	Mean	SD	Mean	SD
1. Agile methodology use	5.01	1.40	4.95	1.43
2. Requirements change	5.04	1.35	5.00	1.30
3. Team size	7.91	1.16	7.80	1.14
4. Programming experience	5.33	2.29	5.27	2.18
5. Project size	491,330	60,200	494,133	61,508
6. Modularity	4.49	1.31	4.42	1.28
7. Bug severity	24.20	10.90	31.04	12.50
8. Component complexity	0.761	0.101	0.610	0.089
9. Coordinative complexity	0.075	0.020	0.105	0.017
10. Dynamic complexity	0.010	0.008	0.022	0.012

Table B3 Descriptive Statistics by Requirements Change

	High requirements change ($n = 28$)		Low requirements change ($n = 28$)	
Variables	Mean	SD	Mean	SD
1. Agile methodology use	5.01	1.40	4.95	1.43
2. Team size	7.90	1.12	7.91	1.15
3. Programming experience	5.33	2.20	5.28	2.26
4. Project size	494,340	62,920	475,089	60,023
5. Modularity	4.40	1.34	4.45	1.29
6. Outcome control	4.48	1.10	4.59	1.13
7. Self control	4.15	1.22	4.20	1.20
8. Bug severity	33.20	11.44	26.38	12.90
9. Component complexity	0.731	0.115	0.601	0.110
10. Coordinative complexity	0.110	0.021	0.068	0.020
11. Dynamic complexity	0.029	0.010	0.010	0.010

References

Aiken, L. S., S. G. West. 1991. *Multiple Regression: Testing and Interpreting Interactions*. Sage, London.

Ang, S., S. A. Slaughter. 2001. Work outcomes and job design for contract *versus* permanent information systems professionals on software development teams. *MIS Quart.* **25**(3) 321–350.

Aoyama, M. 1998. Web-based agile software development. *IEEE Software* **15**(6) 56–65.

Bailyn, L. 1985. Autonomy in the industrial R&D laboratory. *Human Resource Management* **24** 129–146.

Banker, R. D., S. M. Datar, C. F. Kemerer. 1991. A model to evaluate variables impacting productivity on software maintenance projects. *Management Sci.* **37**(1) 1–18.

Banker, R. D., G. B. Davis, S. A. Slaughter. 1998. Software development practices, software complexity, and software maintenance performance: A field study. *Management Sci.* **44**(4) 433–450.

Barki, H., J. Hartwick. 2001. Interpersonal conflict and its management in information system development. *MIS Quart.* **25**(2) 195–228.

Baron, R. M., D. A. Kenny. 1986. The moderator-mediator variable distinction in social psychological research: Conceptual, strategic, and statistical considerations. *J. Personality Soc. Psych.* **51**(6) 1173–1182.

Baskerville, R., L. Levine, J. Pries-Heje, B. Ramesh, S. A. Slaughter. 2002. Balancing quality and agility in Internet speed software development. L. Applegate, R. Galliers, J. I. DeGross, eds. *Proc. 23rd Internat. Conf. Inform. Systems*, Barcelona, Spain, 859–864.

Beath, C. M., W. J. Orlikowski. 1994. The contradictory structure of systems development methodologies: Deconstructing the IS-user relationship in information engineering. *Inform. Systems Res.* **5**(4) 350–377.

Beck, K. 1999. Embracing change with extreme programming. *IEEE Comput.* **32** 70–77.

Beck, K. 2000. *Extreme Programming Explained*. Addison-Wesley, Reading, MA.

Beck, K. 2003. *Test-Driven Development by Example*. Addison-Wesley, Reading, MA.

Bieman, J. 2002. Risks to software quality. *Software Quality J.* **10**(1) 7–9.

Bliese, P. D. 2000. Within-group agreement, non-independence, and reliability: Implications for data aggregation and analysis. K. J. Klein, S. W. J. Kozlowski, eds. *Multilevel Theory, Research, and Methods in Organizations*. Jossey-Bass, San Fransisco, 349–381.

Boehm, B. W. 1981. *Software Engineering Economics*. Prentice-Hall, Upper Saddle River, NJ.

Boehm, B. W. 1991. Software risk management: Principles and practices. *IEEE Software* **8**(1) 32–41.

Boehm, B. W., R. Turner. 2005. Management challenges to implementing agile processes in traditional development organizations. *IEEE Software* **22**(5) 30–39.

Brooks, F. P., Jr. 1987. No silver bullet: Essence and accidents of software engineering. *Computer* **20**(4) 10–19.

Byrd, T. A., D. E. Turner. 2000. Measuring the flexibility of information technology infrastructure: Exploratory analysis of a construct. *J. Management Inform. Systems* **17**(1) 167–208.

Card, D. N. 1992. Designing software for producibility. *J. Systems Software* **17** 219–225.

Card, D. N., R. L. Glass. 1990. *Measuring Software Design Quality*. Prentice-Hall, Englewood Cliffs, NJ.

Carte. T. A., C. J. Russell. 2003. In pursuit of moderation: Nine common errors and their solutions. *MIS Quart.* **27**(3) 479–501.

Chan, D. 1998. Functional relations among constructs in the same content domain at different levels of analysis: A typology of composition models. *J. Appl. Psych.* **83** 234–246.

Choudhury, V., R. Sabherwal. 2003. Portfolios of control in outsourced software development projects. *Inform. Systems Res.* **14**(3) 291–314.

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

Cockburn, A. 2001. *Agile Software Development*. Addison-Wesley, Reading, MA.

Cohen, S. G., D. E. Bailey. 1997. What makes teams work? Group effectiveness research from the shop floor to the executive suite. *J. Management* **23** 239–290.

Cohen, S. G., G. E. Ledford, G. M. Spreitzer. 1996. A predictive model of self-managing work team effectiveness. *Human Relations* **49** 643–676.

Conboy, K. 2009. Agility from first principles: Reconstructing the concept of agility in information systems development. *Inform. Systems Res.* **20**(3) 329–354.

Curtis, B., H. Krasner, N. Iscoe. 1988. A field study of the software design process for large systems. *Comm. ACM* **31**(11) 1268–1287.

Cusumano, M. A., D. B. Yoffie. 1999. Software development on Internet time. *IEEE Comput.* **32**(10) 60–69.

Darcy, D. P., S. A. Slaughter, C. F. Kemerer, J. E. Tomayko. 2005. The structural complexity of software: An empirical test. *IEEE Trans. Software Engrg.* **31**(11) 982–995.

DeVellis, R. F. 2003. *Scale Development: Theory and Applications*. Sage, Thousand Oaks, CA.

Duncan, N. B. 1995. Capturing flexibility of information technology infrastructure: A study of resource characteristics and their measure. *J. Management Inform. Systems* **12**(2) 37–57.

Eisenhardt, K. M. 1985. Control: Organizational and economic approaches. *Management Sci.* **31**(2) 134–149.

Fitzgerald, B. 2000. Systems development methodologies: The problem of tenses. *Inform. Tech. People* **13** 13–22.

Fitzgerald, B., G. Hartnett, K. Conboy. 2006. Customising agile methods to software practices at Intel Shannon. *Eur. J. Inform. Systems* **15** 200–213.

Fowler, M. 2005. The new methodology. Retrieved May 26, 2007, http://www.martinfowler.com/articles/newMethodology.html.

Fowler, M., J. Highsmith. 2001. Agile methodologists agree on something. *Software Development* **9** 28–32.

Gefen, D., M. Keil. 1998. The impact of developer responsiveness on perceptions of usefulness and ease of use: An extension of the technology acceptance model. *DATABASE Adv. Inform. Systems* **29**(2) 35–49.

Gerwin, D., L. Moffat. 1997. Withdrawal of team autonomy during concurrent engineering. *Management Sci.* **43**(9) 1275–1287.

Greene, W. H. 1997. *Econometric Analysis*, 3rd ed. MacMillan Publishing Company, New York.

Guinan, P. J., J. G. Cooprider, S. Faraj. 1998. Enabling software development team performance during requirements definition: A behavioral versus technical approach. *Inform. Systems Res.* **9**(2) 101–125.

Hackman, J. R. 1986. The psychology of self-management in organizations. M. S. Pallack, R. O. Perloff, eds. *Psychology and Work: Productivity, Change, and Employment*. American Psychological Association, Washington, DC, 89–136.

Henderson, J. C., S. Lee. 1992. Managing I/S design teams: A control theories perspective. *Management Sci.* **38**(6) 757–777.

Highsmith, J., A. Cockburn. 2001. Agile software development: The business of innovation. *IEEE Comput.* **34**(9) 120–122.

Hoorn, J. F., E. A. Konijn, H. van Vliet, G. van der Veer. 2007. Requirements change: Fears dictate the must haves; desires the won't haves. *J. Systems Software* **80**(3) 328–355.

Humphrey, W. S. 1995. *A Discipline for Software Engineering*. Addison-Wesley, Reading, MA.

Iansiti, M., A. MacCormack. 1997. Developing products on Internet time. *Harvard Bus. Rev.* **75**(5) 108–117.

James, L. R., R. G. Demaree, G. Wolf. 1984. Estimating within group interrater reliability with and without response bias. *J. Appl. Psych.* **69** 219–229.

Jaworski, B. J. 1988. Toward a theory of marketing control: Environmental context, control types, and consequences. *J. Marketing* **52** 23–39.

Kemerer, C. F. 1995. Software complexity and software maintenance: A survey of empirical research. *Ann. Software Engrg.* **1**(1) 1–22.

Kirkman, B. L., D. L. Shapiro. 2001. The impact of cultural values on job satisfaction and organizational commitment in self-managing work teams: The mediating role of employee resistance. *Acad. Management J.* **44**(3) 557–569.

Kirsch, L. J. 1996. The management of complex tasks in organizations: Controlling the systems development process. *Organ. Sci.* **7**(1) 1–21.

Kirsch, L. J. 1997. Portfolios of control modes and IS project management. *Inform. Systems Res.* **8**(3) 215–239.

Kirsch, L. J. 2004. Deploying common systems globally: The dynamics of control. *Inform. Systems Res.* **15**(4) 374–395.

Kirsch, L. J., V. Sambamurthy, D.-G. Ko, R. L. Purvis. 2002. Controlling information systems development projects: The view from the client. *Management Sci.* **48**(4) 484–498.

Klein, K. J., F. Dansereau, R. J. Hall. 1994. Levels issues in theory development, data collection, and analysis. *Acad. Management Rev.* **19**(2) 195–229.

Langfred, C. W. 2004. Too much of a good thing? Negative effects of high trust and individual autonomy in self-managing teams. *Acad. Management J.* **47** 358–399.

Larman, C. 2003. *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley, Boston, MA.

Lee, G., W. Xia. 2005. The ability of information systems development project teams to respond to business and technology changes: A study of flexibility measures. *Eur. J. Inform. Systems* **14** 75–92.

MacCormack, A., R. Verganti, M. Iansiti. 2001. Developing products on "Internet time": The anatomy of a flexible development process. *Management Sci.* **47**(1) 133–150.

Manz, C. C., H. P. Sims, Jr. 1987. Leading workers to lead themselves: The external leadership of self-managing work teams. *Admin. Sci. Quart.* **32** 106–128.

Maruping, L. M., R. Agarwal. 2004. Managing team interpersonal processes through technology: A task-technology fit perspective. *J. Appl. Psych.* **89**(6) 975–990.

Mathiassen, L., T. Tuunanen, T. Saarinen, M. Rossi. 2007. A contingency model for requirements development. *J. AIS* **8**(11) 569–597.

McCabe, T. J. 1976. A complexity measure. *IEEE Trans. Software Engrg.* **2**(4) 308–320.

Nidumolu, S. 1995. The effect of coordination and uncertainty on software project performance: Residual performance risk as an intervening variable. *Inform. Systems Res.* **6**(3) 191–219.

Nidumolu, S. R., M. Subramani. 2003. The matrix of control: Combining process and structure approaches to managing software development. *J. Management Inform. Systems* **20**(3) 159–196.

Nosek, J. 1998. The case for collaborative programming. *Comm. ACM* **41**(3) 105–108.

Ouchi, W. G. 1979. A conceptual framework for the design of organizational control mechanisms. *Management Sci.* **25**(9) 833–848.

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

Poppendeick, M. 2001. Lean programming. *Software Development* **9** 71–75.

Rising, L., N. S. Janoff. 2000. The Scrum software development process for small teams. *IEEE Software* **17**(4) 26–32.

Ryan, T. Y. 1997. *Modern Regression Analysis*. Wiley, New York.

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

Sillince, J. A. A., S. Mouakket. 1997. Varieties of political process during systems development. *Inform. Systems Res.* **8**(4) 368–397.

Slaughter, S. A., L. Levine, R. Balasubramaniam, J. Pries-Heje, R. Baskerville. 2006. Aligning software processes with strategy. *MIS Quart.* **30**(4) 891–918.

Standish Group. 2003. Chaos report. Accessed June 6, 2006, http://www. standishgroup.com.

Straub, D. W. 1989. Validating instruments in MIS research. *MIS Quart.* **13**(2) 147–169.

Vessey, I. 1989. Toward a theory of computer program bugs: An empirical test. *Internat. J. Man-Machine Stud.* **30**(3) 23–46.

Wageman, R. 1995. Interdependence and group effectiveness. *Admin. Sci. Quart.* **40** 145–180.

Walz, D. B., J. J. Elam, B. Curtis. 1993. Inside a software design team: Knowledge acquisition, sharing, and integration. *Comm. ACM* **36**(10) 63–77.

Weinberg, G. 1971. *The Psychology of Computer Programming*. Van Nostrand Reinhold, New York.

Zellner, A. 1962. An efficient method for estimating seemingly unrelated regressions and tests for aggregation bias. *J. Amer. Statist. Assoc.* **57** 348–368.