ABSTRACT

Title of dissertation:	ESSAYS ON AGILITY IN SOFTWARE DEVELOPMENT TEAMS: PROCESS AND GOVERNANCE PERSPECTIVES
	Likoebe Mohau Maruping, Doctor of Philosophy, 2006
Dissertation directed by:	Professor Ritu Agarwal Decision and Information Technologies
	Professor Viswanath Venkatesh Information Systems Department University of Arkansas

Software project teams are often faced with the potential for changing requirements during software development. Such changes present a significant challenge to software project teams often resulting in cost overruns, missed deadlines, and faulty code. While the phenomenon of requirements changes by itself is not new, the frequency with which such changes occur has increased in recent years. Software project teams continue to be challenged by the uncertainty caused by the potential for such changes. Flexibility has been identified as a desirable characteristic of teams in this type of environment. This dissertation is made up of two essays that examine the processes and governance mechanisms that can potentially enable software project teams to achieve greater flexibility. The first essay provides a team process-oriented view of the problem. Specifically, agile programming practices and boundary spanning activities are argued to enhance software project team effectiveness by receiving customer feedback and incorporating requirements changes on an on-going basis. A longitudinal field study of 56 software development teams provides support for these arguments. The second essay adopts a team governance lens. Specifically, formal and informal control modes are posited to influence software project team effectiveness. Control mechanisms that promote team autonomy and flexibility are argued to positively influence software project team effectiveness. Further, team leader behaviors are also expected to play a critical role in facilitating software project team effectiveness. These arguments are tested and largely supported. Together, the two essays shed light on how software project teams can be designed to have greater flexibility—an important stepping stone for managing software development in the face of requirements uncertainty.

ESSAYS ON AGILITY IN SOFTWARE DEVELOPMENT TEAMS: PROCESS AND GOVERNANCE PERSPECTIVES

by

Likoebe Mohau Maruping

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2006

Acknowledgements

There are numerous people to acknowledge in this dissertation:

I would like to begin by thanking my dissertation co-chairs Dr. Viswanath Venkatesh and Dr. Ritu Agarwal for their guidance and unwavering support during the process of developing this dissertation. I am grateful for the significant amount of time they devoted to ensuring that this dissertation was successfully completed. I am also thankful for their patience during times when I was not at my best. Dr. Venkatesh and Dr. Agarwal have not only been excellent advisors, but also incredible friends as well. I learned so much about research and academia from you. Many thanks for everything!

My dissertation also benefited immensely from my committee members Dr. Katherine Stewart, Dr. Paul Tesluk, and Dr. Paul Hanges. Thanks for the encouragement, support, and feedback that made this dissertation what it is. Dr. Stewart and Dr. Tesluk have not only been valuable committee members, but good friends as well.

I would also like to thank the faculty in the Decision and Information Technologies department at the University of Maryland. I learned a lot about research from them, both through doctoral seminars and through informal conversations.

A debt of gratitude goes to my fellow doctoral students who were with me during this journey. Especially: Riki Takeuchi, Tashfeen Sohail, Fei Ye, Qing Cao, Animesh, Jason Kuruzovich, and Steven Johnson. There are many other fellow PhD students who made the doctoral program a joy to be a part of. Thanks for the support and the laughs that were needed during the tough times.

Thanks to my friends outside of academia, all of whom I've known prior to my entry into the PhD program. Especially my closest friends: Kevin Bloomfield, Kenneth Boda, Amanda Boda, and Ming Wei Ong. Thanks for the constant support and their patience when I have failed at keeping in touch. They have stood by me all the same. No one could ask for better friends.

Words cannot express my gratitude to my family for what they have done to get me to this point. I am grateful to my father Dr. Mothae Maruping for sparking my interest in academia and to my mother Dr. Arabang Maruping for her support. Thanks to my younger sisters Motselisi, Mpho, and Nkabeng Maruping for everything. We have been through a lot together. In many ways the completion of this degree is tribute to all of us.

Finally, I would like to thank my partner and soul mate Miyuki Maruping. She has been an incredible cheerleader since my days as an undergraduate student. Miyuki has been my pillar of strength throughout this process, extending her support even across the Pacific Ocean. She made numerous sacrifices to ensure I completed this degree, including postponing our honeymoon. Thanks for the encouragement during this entire process. I am also thankful to Miyuki's family (now my family too): Mr. Masahide Chiba, Mrs. Yuriko Chiba, and Mr. Hideki Chiba for their support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

Overview

For decades, software developers have wrestled with the challenge of curbing the level of complexity in software development projects while also providing all of the functionality required by users (Reifer 2000). As complexity increases, software code becomes more expensive to change (Banker and Slaughter 2000). Such changes often result in missed delivery deadlines and in the worst case, project failure. The challenge is further exacerbated when requirements change during the software development process. The costs of implementing changes to software code increase exponentially as the project progresses (Pfleeger 1987). In fact, the cost of changing software code in response to changing requirements contributes to an estimated $17 billion in project cost overruns in the U.S. alone (Standish Group 2003). Iansiti and MacCormack (1997) have suggested that the requirements for projects are changing more rapidly and more frequently than in the past. Unfortunately, as practitioners and academics alike have argued, traditional software development methodologies are ill equipped to manage such changes (Fowler and Highsmith 2001; Highsmith 1999). A report by the Standish Group indicates that in 2003 only 52% of required features and functions in software made it to the final software product release (Standish Group 2003). Thus, the business community has called for lightweight software accompanied by faster and more flexible software development processes (Fowler and Highsmith 2001). Agile methodologies have been proposed as a solution to manage these challenges (Highsmith 2002; McCauley 2001).

In light of the growing pressures in the Internet software development environment, a number of methodologies have emerged within the developer community.

Collectively these methodologies are referred to as *agile methodologies*. Agile methods emphasize speed and simplicity in software development work (Beck 1999; McCauley 2001). Development teams achieve such agility by focusing on delivering immediately required functionalities as quickly as possible, continuously receiving customer feedback, and reacting to changing requirements (Beck 1999). While a number of agile methodologies have emerged over the past few years, one particular methodology, extreme programming, has received a significant amount of attention in both industry and academia.

Extreme programming is characterized by a set of development practices that are geared towards enabling small teams (of no more than 10 programmers) to successfully develop software while faced with the challenge of vague and continually changing requirements (Beck 2000). The philosophy underlying extreme programming is constructed around four core values: communication, simplicity, feedback, and courage. The pragmatic manifestation of these core values is encapsulated in a set of practices that are collectively referred to as the circle of life (Beck 2000). Each of the practices examined in this dissertation is briefly described in Table 1.1.

The adoption and implementation of extreme programming practices has yielded some remarkable success stories. For instance, Poole and Huisman (2001) tracked the successful adoption of several extreme programming practices at Iona Technologies. The adoption of extreme programming practices was prompted by problems with existing: processes and practices, testing, code entropy, and team morale (Poole and Huisman 2001). The successful adoption of extreme programming practices led Iona Technologies to realize productivity gains, reduced defect rates, and improved team morale.

DaimlerChrysler and Ford Motor Company have documented similar extreme programming success cases (Beck 1999).

Table 1.1: Agile Programming Practices.

Agile Practice	Description
Planning	The scope of the next release is determined through a combination of business priorities and technical estimates. Scope is outlined by customers, while developers track progress.
Testing	Programmers write unit tests while customers or other programmers write acceptance tests for the code. The project does not move forward until the unit tests run flawlessly.
Refactoring	One of the programmers' daily activities is improving the design of existing code without changing its behavior. This practice involves the elimination of redundancies and the reduction in complexity in the code.
Pair programming	During the coding process two developers work side by side on a single computer. Typically one programmer codes while the other checks the quality of the code (these roles are rotated over time).
Continuous integration	This is a process in which programmers load the current release of the code and integrate the most recent set of code. The newly integrated code is then tested for errors.

Although extreme programming practices (henceforth referred to as agile programming practices) continue to be widely adopted across organizations, they are rarely adopted in pure form. In other words, there is a fair amount of variation across software development teams in the specific practices that are adopted as well as the extent to which adopted practices are utilized. Hence, it is unclear which of these practices contribute to software development team effectiveness. Further, not much is known about the precise mechanisms through which these practices affect effectiveness. Finally, little is known about the team conditions that enhance or inhibit the effectiveness of these practices. The aim of this dissertation is to provide an in-depth examination of the team dynamics in software development teams employing agile software development practices. The dissertation is geared towards developing a theoretical understanding of *how* agile programming practices influence software development teams' ability to effectively manage software projects. Such practices could yield important insights into how to manage the uncertainty caused by frequently changing requirements in software projects. In order to accomplish this objective, I examine software development teams in two essays. The two essays are motivated by separate but related research questions.

To provide a rich understanding of software development teams and their use of agile programming practices, a dual focus is adopted in the essays that follow. The focus of the first essay is on the internal processes of software development teams. Specifically, the first essay draws links between software development teams' use of agile programming practices and team effectiveness. Drawing on the boundary spanning literature, the essay highlights team activities that can enhance the effectiveness of specific agile programming practices. In contrast to the first essay, the second essay focuses on the team's leadership and the control mechanisms used to manage the team's progress toward project completion. Drawing on control theory, the second essay examines how various team governance mechanisms enable or constrain software development teams' ability to be effective. An increasing emphasis on the importance of understanding phenomena at various levels of analysis has spawned several calls for more research to develop multilevel theories and conduct multilevel empirical investigations (Klein and Kozlowski 2000). Thus, overall, the dissertation provides a multilevel view of software development teams and their use of agile programming practices. The essays that make up this dissertation are briefly outlined next.

Essay 1 is focused on team boundary-spanning process (Ancona and Caldwell 1992; Gladstein 1984) and its influence on the effectiveness of activities enacted in various phases of software development team work. While prior research has identified key project-related determinants of software development performance in dynamic environments (e.g., Bhattacharya, Krishnan and Mahajan 1998; MacCormack, Verganti and Iansiti 2001), the interpersonal interactions among members of the software development team has been largely treated as a blackbox. MacCormack et al. (2001) found customer feedback to be one of the key determinants of project performance. Similarly, Bhattacharya et al. (1998) highlighted the importance of feedback in product development under conditions of uncertainty. However, the precise mechanisms through which interaction with the external environment enables these teams to enhance their performance has not been explored within this literature. Essay 1 is aimed at opening up this blackbox and examining the agile programming practices and boundary-spanning activities that contribute to software development team effectiveness. I examine how boundary-spanning influences the effectiveness of software development team task-focused (Guinan, Cooprider and Faraj 1998) and decision-making (Eisenhardt 1989; Simons, Pelled and Smith 1999) processes.

Essay 2 is aimed at understanding how team leadership and team governance enable or constrain software development team effectiveness. Prior research has suggested that the exercise of control in software development projects is an effective mechanism that managers can use to ensure that organizational objectives are met (Henderson and Lee 1992; Kirsch 1997). However, agile programming practices are designed to manage uncertainty. Many of these practices possess characteristics of activities observed in self-managing teams where autonomy is especially important (Manz and Sims 1987). An inherent tension therefore exists between the autonomy needed by teams using these programming practices and the need for control and structure in software development projects. Drawing on research on control theory for software development projects (e.g., Kirsch 1997; Kirsch, Sambamurthy, Ko and Purvis 2002) as well as research on leadership of self-managing teams, I develop a model to understand and resolve this tension. While research on the effective management of software development projects has established a rich tradition, very little research has examined project leadership under conditions of high uncertainty, particularly in teams engaging in activities designed to manage such conditions. Hence, a need exists, to understand what it is that effective leaders of such projects do.

In sum, this dissertation is aimed at providing a multilevel view of team dynamics and teams' use of agile programming practices in a software development environment. The dissertation will contribute to the literature in several ways. The first essay will contribute to the IS software development literature through its examination of the mechanisms through which customer feedback improves software development team effectiveness. The essay will outline specific mechanisms through which customer feedback can be used to manage changes in requirements. This is particularly important because it will allow researchers to examine the phenomenon at a finer level of granularity. Second, the model developed will be rooted in team action. The model builds on specific activities/practices that developers engage in, that are designed to enhance software development teams' agility in responding to changes in requirements. Thus, specific interventions can be developed to enhance software projects' adaptability to changes in requirements, by implementing the practices outlined in the essay. Further, the effects of these practices can be tested. Finally, the second essay in the dissertation will contribute to the IS literature on control theory. The impact of various control modes on affective team outcomes has not received a lot of attention in the IS literature. Yet these affective outcomes are critical given their implications for team effectiveness (Marks, Mathieu and Zaccaro 2001). Second, specific leader behaviors that can enhance software project team effectiveness under such conditions have not been identified or examined in the literature to date. The identification of such behaviors would give team leaders specific interventions that may boost software project team effectiveness. Overall, the essay would contribute to the literature by examining the mix of control modes and team leader behaviors that most effectively enhance software project team effectiveness and affective team outcomes.

In the next section, I develop each of the essays and develop specific hypotheses to be tested in the dissertation. For each essay, I describe the methodology used to test the hypotheses, including data collection and analysis. Finally, I conclude with a discussion of the results and the implications for future research and practice.

CHAPTER II

ESSAY 1: AGILE SOFTWARE DEVELOPMENT PRACTICES, CUSTOMER FEEDBACK, AND SOFTWARE PROJECT TEAM EFFECTIVENESS: A PRACTICE-ORIENTED PERSPECTIVE

Abstract

The challenge of managing changing software project requirements has endured for decades. Recent literature suggests that software project requirements are changing with greater frequency. Unfortunately, traditional approaches to software project work have proved to be limited in addressing these challenges. Agile methodologies have emerged as a response to the challenge of changing requirements. Organizational software project teams are beginning to adopt these agile methodologies. Although anecdotal evidence highlights the promise of these methodologies, little empirical evidence exists to support their effectiveness. Further, little research has sought to explicate the precise mechanisms through which customer feedback—which is important in uncertain environments—actually impacts software project team effectiveness. Drawing on the boundary spanning literature, this research examines the link between agile programming practices, boundary spanning activities, and software project team effectiveness. Using a field sample of 56 software development teams, I find that task-focused and decision-making agile programming practices explain 19.3% of the variance in leader-rated team effectiveness and 11.8% of the variance in team-rated effectiveness. Boundary spanning is found to play a significant moderating role in predicting software project team effectiveness. The implications of these findings for theory and practice are discussed.

Introduction

For decades, software developers have wrestled with the challenge of curbing the level of complexity in software development projects while also providing all of the functionality required by users (Reifer 2000). As complexity increases, software code becomes more expensive to change (Banker and Slaughter 2000). Such changes often result in missed delivery deadlines and in the worst case, project failure. The challenge is further exacerbated when requirements change during the software development process. The costs of implementing changes to software code increase exponentially as the project progresses (Pfleeger 1987). In fact, the cost of changing software code in response to changing requirements contributes to an estimated $17 billion in project cost overruns in the U.S. alone (Standish Group 2003). Prior studies of software development have shown that changes in software design and functionality are more costly to implement at later stages of the development process (e.g., Banker and Slaughter 2000; Harter, Krishnan and Slaughter 2000). Iansiti and MacCormack (1997) have suggested that the requirements for projects are changing more rapidly and more frequently than in the past. Consequently, the current environment for software development is marked by high degrees of uncertainty (MacCormack, Verganti and Iansiti 2001). Unfortunately, as practitioners and academics alike have argued, traditional software development methodologies are ill equipped to manage such changes (Fowler and Highsmith 2001; Highsmith 1999). As noted earlier, a report by the Standish Group indicates that in 2003 only 52% of required features and functions in software made it to the final software product release (Standish Group 2003). Agile methodologies have been proposed as a solution to manage these challenges (Highsmith 2002; McCauley 2001).

Much of the research on software development under uncertainty has focused on various characteristics/features of the software development process that contribute to overall software project performance. For instance, MacCormack, Verganti, and Iansiti (2001) studied the determinants of software project performance under conditions of uncertainty and found that customer feedback during the software development process was a key predictor. Similarly, Bhattacharya, Krishnan, and Mahajan (1998) argued that receiving feedback from end users was a fundamental predictor of software project performance under conditions of uncertainty.

Although prior literature has highlighted the importance of factors such as customer feedback, the actual activities that developers engage in during the software development process have been largely treated as a blackbox. Developers on software project teams are the individuals responsible for the actual coding process. Additionally, developers are responsible for ensuring that the software code remains adaptable to uncertainty, including the possibility of future changes in requirements. Of particular importance are those practices that developers engage in that help to reduce the costs of changing code during later stages of development. Anecdotal evidence suggests that several agile software development practices have successfully helped software development teams to manage uncertainty (e.g., Beck 1999; Murru, Deias and Mugheddu 2003; Poole and Huisman 2001). However, there is no empirical evidence of the effectiveness of these practices. Further, an extensive search indicated that no scales currently exist, to measure the use of these practices.

The purpose of this essay is to identify how teams using agile programming practices can enhance their effectiveness by gaining and incorporating timely requirement information into ongoing software projects. *Team effectiveness* is defined as the quality of the team's work on a project (Cohen and Bailey 1997). Research on new product development, which involves high levels of uncertainty, highlights boundary spanning as a key determinant of success for project teams (e.g., Ancona and Caldwell 1992; Tushman 1977). The team boundary spanning literature addresses the concept of acquiring timely information from the environment and, thus, serves as the theoretical basis for this essay. Drawing from existing research on team boundary spanning, a model focused on team process is developed. The model is geared towards task accomplishment in teams using agile programming practices to manage software development projects.

A set of common characteristics set agile methodologies apart from traditional software development processes; they include: (1) a deliberate focus on people rather than processes and tools; (2) the pursuit of adaptation as opposed to optimization; (3) an emphasis on working software of a high quality rather than documentation; (4) establishing and maintaining continuous customer involvement during development; (5) maintaining flexibility and being open to change; and (6) using short iterations of planning, organizing, and coding over the life of the project (Beck 2000; Cockburn 2002). These characteristics highlight the importance of examining interpersonal team processes during software development.

Marks et al. (2001) suggest that all project teams necessarily go through two phases for task accomplishment: a planning (transition) phase and an action phase. Accordingly, this essay focuses on two specific team processes: task-focused activities (which correspond to the action phase) and decision-making processes (which correspond to the planning phase). *Task-focused activities* are the means by which teams coordinate the flow and use of information and expertise toward task accomplishment (Faraj and Sproull 2000). *Decision-making processes* are the mechanism through which teams use available information to determine the most appropriate actions to be taken towards task accomplishment (Eisenhardt 1989; Schweiger, Sandberg and Rechner 1989). Software development teams draw heavily on task-focused and decision-making activities during the software development process. Specific programming expertise is needed to code various functionalities into the software product (Curtis, Krasner and Iscoe 1988; Walz, Elam and Curtis 1993). Further, the coordination of specific expertise is necessary to ensure that the various components of the software are integrated such that the entire system works optimally (Nidumolu 1995). Similarly, in order to successfully manage the software development process, teams need to make important decisions regarding the functionalities that need to be embedded in the next release of the software (Beck 1999). The rapid pace and uncertainty that characterize the current environment in which software project teams are embedded make it extremely important that these team processes be directed towards meeting customer requirements in a timely manner (MacCormack et al. 2001). Getting customers actively involved in information exchange during the development process ensures that the team is continuously apprised of current requirements. Hence, boundary spanning is an important process, which can enhance the effectiveness of the task-focused and decision-making processes that software project teams engage in. Contrary to previous research (e.g., Bhattacharya et al. 1998; MacCormack et al. 2001), in this essay, I argue that boundary spanning moderates the relationship between task-focused and decision-making processes, and software project team effectiveness through the solicitation of customer feedback on project requirements and customer involvement in decision-making. Information acquired from the environment is used in task-focused and decision-making processes that spur team effectiveness at accomplishing tasks.

In sum, the key objectives in this essay are to:

- Define and describe the key task-focused agile programming practices;
- Draw on the boundary spanning literature to understand how customer feedback contributes to software project team effectiveness;
- Develop scales to measure the agile practices; and
- Empirically test the proposed relationships.

In the next section, I review the team boundary spanning literature. I then synthesize this literature along with the agile software development literature to develop hypotheses around the intersection of team task-focused (agile programming practices) and decision-making processes, and software project team effectiveness. Finally, the scale development process and empirical testing of the proposed model is described, followed by a discussion of the implications of the findings for research and practice.

Theoretical Background

The management of team boundaries has been identified as an important team activity in ongoing work teams. *Boundary spanning* is conceptualized as the team's management of relationships with other entities, who provide inputs to or receive outputs from the group (Gladstein 1984; Katz and Kahn 1978; Tushman 1977). Groups manage their boundaries by interacting and communicating with external stakeholders. Within organizations, boundary spanning involves maintaining the visibility of the team through on going communication, ensuring that managers understand the mission of the team, and making sure that necessary resources are available. For instance, teams can manage their access to resources by communicating the importance of the team and its mission as well as the need for ongoing organizational support (Ancona and Caldwell 1992; Druskat and Wheeler 2003). Teams also need to ensure that their outputs meet the requirements of their customers. Project teams can accomplish this by communicating with customers and assessing their requirements (Cohen and Bailey 1997).

Teams often exist and operate within an organizational context (Gladstein 1984). The extant literature emphasizes the environment as an important determinant of team effectiveness (Guzzo and Shea 1992; Hackman 1987). The environment possesses important information about potential facilitators and/or impediments to effective task accomplishment. Maintaining an awareness of the environment enables teams to adapt their internal processes. Workarounds can be developed in the face of potential impediments to task accomplishment (Tesluk and Mathieu 1999). Teams can also capitalize on the discovery of resources or factors that facilitate task performance. Customer satisfaction—an important outcome of teamwork (Cohen and Bailey 1997)—is accomplished when team outputs meet specified customer requirements. Hence, it is important that teams develop and maintain an up-to-date understanding of the customer's needs. Keeping apprised of customer needs can be accomplished by engaging in ongoing communication with customers (Ancona and Caldwell 1992).

Research in the organizational literature has examined the influence of boundary spanning on team performance in a variety of contexts. In a study of self-managing teams in a Fortune 500 manufacturing plant, Druskat and Wheeler (2003) found that the most effective teams were those that engaged in boundary spanning activities. Specifically, they found that effective teams had a leader who was actively involved in garnering external support for their team's purpose and activities. Boundary spanning behaviors included persuading managers and other constituents to provide assistance and resources (Druskat and Wheeler 2003). Ancona (1990) examined the strategies used by teams to manage the external environment. Her study of five consulting teams revealed three strategies that were employed: informing, parading, and probing. The study results indicated that teams that utilized the probing strategy had the highest performance. The probing strategy involved the team's active interaction with outsiders so as to revise and update their map of the environment (Ancona 1990). Ancona and Caldwell (1992) conducted a longitudinal study of 45 new-product teams. They identified three categories of externally oriented activities that teams engage in within their work environment. Ambassadorial activities included protecting the team from outside pressure, lobbying for resources, and persuading others to support the team. Task coordination activities encompassed functions such as getting feedback on product design and negotiating with entities that were external to the group. The scouting activity was characterized as those activities related to information gathering (Ancona and Caldwell 1992). Ambassadorial and task coordination activities were found to contribute positively to team performance while scouting activities negatively influenced performance (Ancona and Caldwell 1992).

Boundary Spanning as Information Gathering

Within the context of the current research, customer feedback has been highlighted as one of the key predictors of new product development success (Ancona and Caldwell 1992; MacCormack et al. 2001). Such feedback comes in the form of specific information regarding customer preferences and requirements. It is through gathering information from the external environment that teams are able to understand their purpose (goals), context (available resources), and progress. The team boundary spanning literature broadly categorizes externally oriented activities into those directed towards the organization and those directed towards customers (whether they be internal or external to the organization within which the team is embedded). Boundary spanning activities that are directed toward the organization involve seeking information from managers, protecting the team from outside pressure, lobbying for resources, and securing managerial support for team initiatives (Ancona and Caldwell 1992; Druskat and Wheeler 2003). Ancona and Caldwell (1992) labeled activities of this form *ambassador* activity. Druskat and Wheeler (2003) called such behaviors *persuading* activities. Ancona (1990) defined *parading* as behaviors in which teams communicated their goals and purpose to management. Teams engaging in ambassador activities frequently communicate with entities higher up in the hierarchy, including division and even corporate management (Ancona and Caldwell 1992).

In contrast to management-oriented boundary spanning behaviors, client-oriented boundary spanning behaviors emphasize information gathering. Teams initiate communication with customers in an effort to acquire information that will help improve team process output. Boundary spanning behaviors of this type also have different labels in the literature. For instance, Ancona (1990) defined *probing* as a team boundary spanning strategy that stressed "external processes, requiring team members to have a lot of interaction with outsiders to diagnose their needs and experiment with solutions" (p. 344). According to Druskat and Wheeler (2003) *scouting* involves the search for information in order to identify organizational needs (where the organization is the client). Scouting involves such behaviors as seeking information from specialists and investigating problems systematically (Druskat and Wheeler 2003). Finally, Ancona and Caldwell (1992) define scouting and *task coordination* as boundary spanning activities directed towards gathering information from outsiders to assess their needs. The scouting activity in Ancona and Caldwell's (1992) conceptualization represents activities geared towards gathering information about the market and competitors rather than being customer-need focused. These activities represent team-initiated interaction with customers to obtain information, which is subsequently used toward team goal accomplishment. Given the amount of overlap in the various conceptualizations of information gathering boundary spanning activities, I use the term *customer feedback* and adopt Ancona and Caldwell's (1992) definition of the construct.

As I just noted, customer feedback is focused on information gathering activities such as requesting feedback reports from customers. There is an additional mechanism for gathering information, which demands tighter coupling between teams and their customers. With tight coupling, the customer is actively involved in ongoing team activities. For example, extreme programming teams often have a customer conduct unit tests during various iterations of the software project (Beck 1999). Activities such as joint application development involve the active participation of users in the software development process. This form of coupling between software project teams and their customers represents another form of boundary spanning, as related to information gathering. Therefore, *customer participation* is defined as the extent to which the team involves the customer in its project-related activities.

Research in the information systems literature has emphasized the importance of user participation in the information system development (ISD) process (Barki and Hartwick 1994; Hartwick and Barki 1994). Ives and Olson (1984) reviewed the literature examining the relationship between user involvement in systems development and performance ratings of the system. Their review indicated some evidence of a link between user involvement and performance in terms of system quality and user attitude toward the system. Hunton and Beeler (1997) studied the effect of different levels of user participation in systems development on performance. They found a significant positive relationship between user participation and performance. More recently, Guinan et al. (1998) found that boundary spanning activities, such as visionary processes, positively influenced software development team performance. Visionary processes include scouting and interacting with the external environment. These findings emphasize how important it is that project teams actively seek to engage potential users in the system development process. These studies also provide empirical evidence of the relationship between user participation in the development process and the outcome of the project. More generally, the studies highlight the importance of boundary spanning to software project team effectiveness. However, deeper theorizing is needed to understand *how* boundary spanning activities enable software project teams to perform well.

Research Questions

As mentioned earlier, the core practices that define agile programming include planning, small releases, the use of metaphor, unit testing, refactoring, continuous integration, collective ownership, coding standards, 40-hour work week, pair programming, simple design, and having an on-site customer. While anecdotal evidence suggests that these practices positively contribute to software project team performance, they do not all necessarily require timely information from the external environment in order to be performed effectively. Boundary spanning encompasses team-initiated interaction with the external environment (Ancona 1990). However, given the nascent stage of current research on the use of agile practices in software project teams, it is unclear which practices are affected by customer feedback and customer participation boundary spanning activities.

Research Question 1: Which set of agile programming practices are expected to be more or less effective depending on the extent to which the team engages in different boundary spanning activities?

Many of the practices espoused in agile programming are guided by the exchange of information and expertise. Software developers on the team must use their expertise to solve problems and deliver software functionalities that meet client needs (Faraj and Sproull 2000). Information about requirements enables developers to direct their efforts toward embedding functionalities that are needed by the client and understanding how the final product as a whole should function. Several of the agile programming practices can be broadly categorized as being task-focused or decision-making related processes. It is these task-focused and decision-making related practices that can be affected by the extent to which the team solicits information from the external environment. In light of prior findings that suggest the central role of customer feedback in dynamic and uncertain environments (e.g., MacCormack et al. 2001), it is important to understand how boundary spanning affects the effectiveness of these practices.

Research Question 2: What is the impact of boundary spanning on the effectiveness of task-focused and decision-making processes in software project teams?

Hypothesis Development

Figure 2.1 illustrates the conceptual model in which boundary-spanning activities moderate the relationships between task-focused processes, decision-making processes, and project outcomes. As noted earlier, agile programming practices differ in the extent to which they involve the exchange and use of information. Agile programming practices that are expected to rely extensively on information and individual expertise include: pair programming, continuous integration, refactoring, and planning. For instance, continuous integration, which involves the integration of changes or modifications from unit tests, requires an understanding of the nature of changes in the software code and how those changes fit in with the rest of the project. Refactoring also requires knowledge of the project and its components, as it involves the optimization of code design without altering the behavior of the overall software product (Beck 2000). On the other hand, other agile programming practices can be characterized as outcomes or philosophies that guide the software development process. Collective ownership, design simplicity, coding standards, small releases, and 40-hour work weeks might be considered philosophies about how the software project team wants to manage the software development process, whereas having an on-site customer is a team design feature.

Boundary spanning activities primarily involve teams' interaction with the external environment so as to maintain an awareness of their context. The extent to which software project teams engage in such activities is expected to have an impact on practices related to task accomplishment and decision-making.

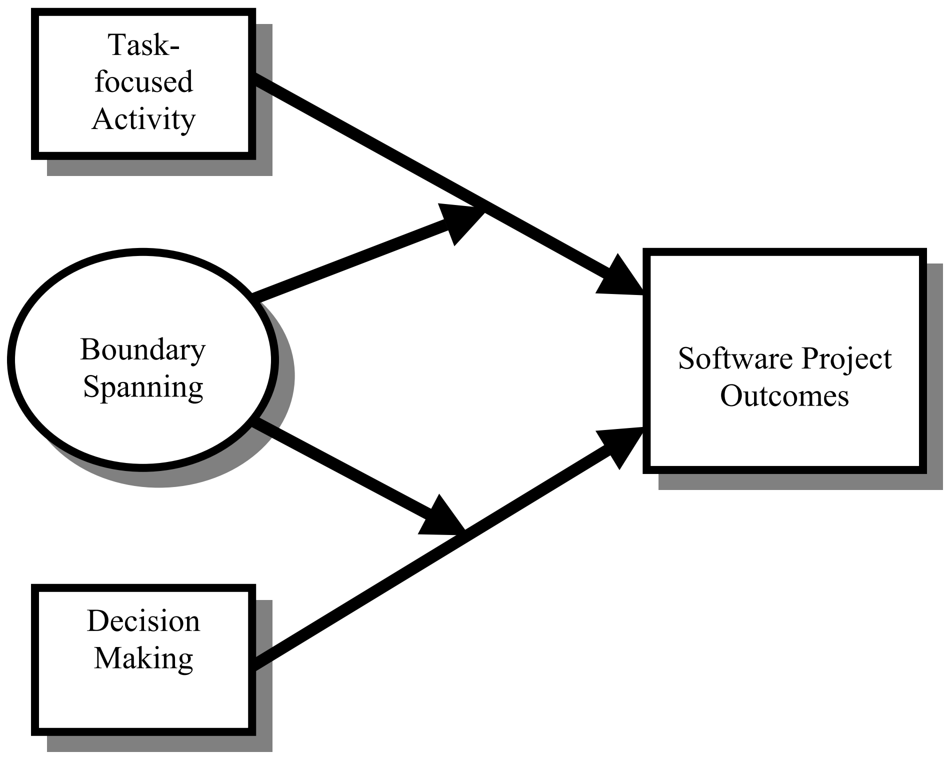


Figure 2.1: Conceptual Model of Software Project Team Process

In this section, I define and describe the agile task-focused and decision-making practices in the model. Specific hypotheses are developed, linking these practices to software project performance. Finally, I develop the logic behind the moderating role of boundary spanning in the relationship between agile practices and software development project team effectiveness. The research model is presented in Figure 2.2.

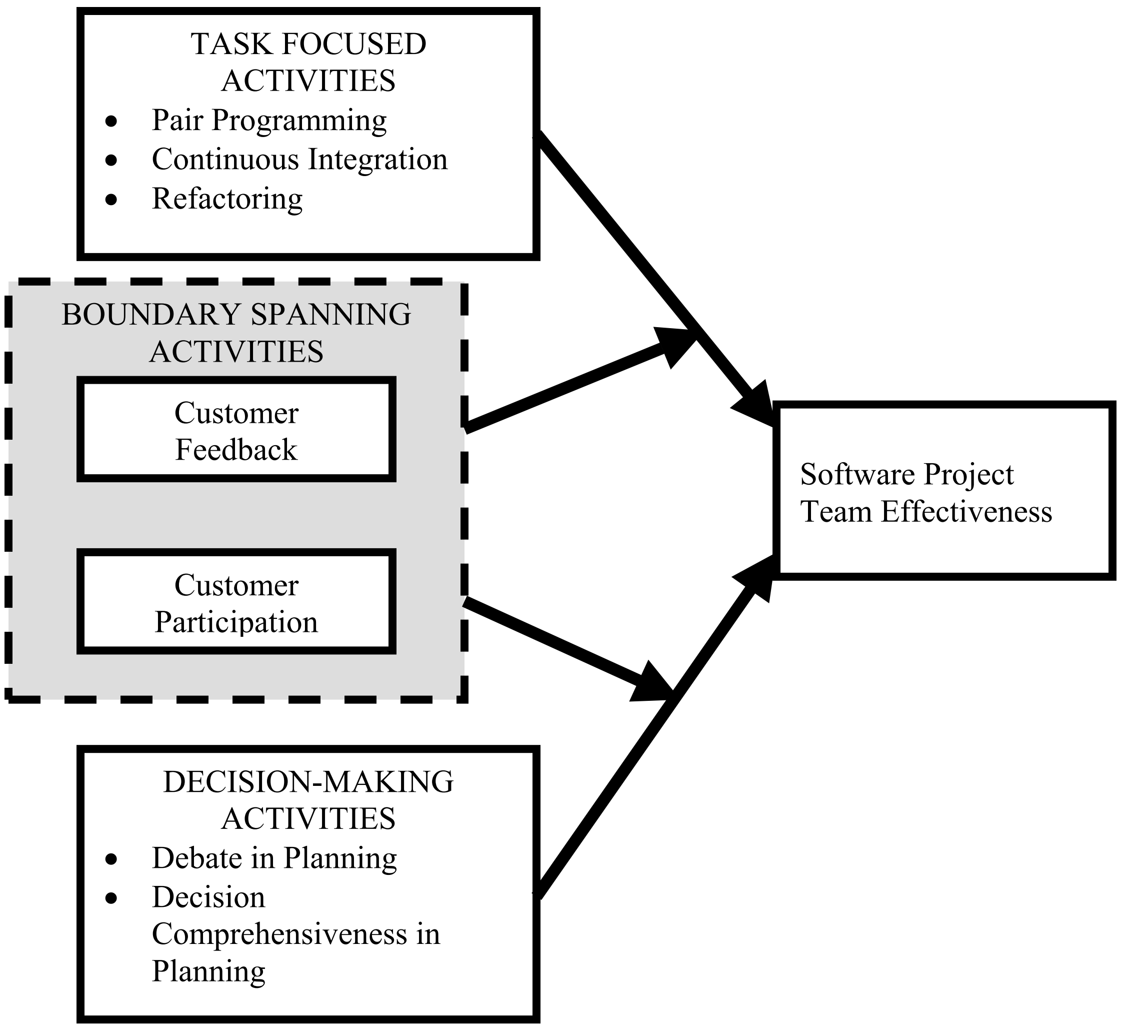


Figure 2.2: Research Model

Task-Focused Activities

Pair programming. The notion that two heads are better than one is captured in the practice of programming in pairs (Williams and Kessler 2000). Collaborative programming involves software development by two developers; with one developer writing code while the other monitors the quality of the code and provides strategic direction for the coding session (Beck 2000; Nosek 1998). The use of this approach to software coding has several benefits. First, it reduces the cognitive load placed on the individual developer to simultaneously develop and monitor the code. While one developer is actively coding, the other is continuously reviewing the analysis, design, and code (Beck 1999). These roles are often switched between partners to provide some task variety. Further, the observing software developer can maintain a "big picture" view of the software coding process and where it fits within the context of the overall project. The coder can therefore devote more cognitive resources to developing software code.

A second benefit to having two minds working on the same problem, is that while there may be a considerable amount of overlap in each partner's knowledge and skill set, there are also likely to be unique skills and expertise that each possess. Hence, through intense continuous interaction, pair programmers are able to pool their resources and make progress toward task accomplishment (Hargrove 1988). In a survey of 750 programmers, Kraut and Streeter (1995) found that one of the most beneficial communication techniques among individual developers was discussion with their peers. Peers are often able to bring a new perspective to problems a programmer is facing (Kraut and Streeter 1995). Pair programming ensures that peers are always available and provides the opportunity for both developers to bring their complementary skills to bear on challenges that may arise during the coding process. In an experiment involving experienced programmers, Nosek (1998) found that five pairs of programmers working on an important organizational task outperformed five individual programmers working on the same task. The pair programmers produced statistically significantly better quality algorithms in less time than the individual programmers (Nosek 1998).

An additional benefit of pair programming is that an additional pair of eyes can more easily identify coding errors during the coding process. Programmers are highly prone to overlooking errors that might be obvious to others (Weinberg 1998). A survey of pair programmers found that a majority of defects in software code could be traced back to components that were written by individuals working alone (Williams and Kessler 2000). Reviewing code and design by pair programming partners helps reduce the likelihood of such defects being overlooked. Constantine (1995) notes that pair programmers often produced code that was close to 100% bug free. Prior research has argued that these factors contribute positively to software project performance (e.g., Nosek 1998; Williams, Kessler, Cunningham and Jeffries 2000). I, therefore, expect the use of pair programming to positively influence software project team effectiveness.

Hypothesis 1a: *The use of pair programming will positively influence software project team effectiveness*.

The client member of a project can add value to the pair programming process in two important ways. First, providing ongoing feedback to paired software developers ensures that the team's development efforts are properly directed towards meeting the customer's needs. Pair programmers can obtain feedback from customers through customer feedback activities (Ancona 1990; Druskat and Wheeler 2003). Customer feedback activities involve high levels of interaction with external groups, obtaining feedback on project design, and getting and coordinating information (Ancona and Caldwell 1992). Given the frequency and speed with which requirements can change, it is important that developers remain aware of the current needs of the client. MacCormack et al. (2001) emphasize the importance of having a feedback mechanism on a product's system-level performance to ensure that the final product better meets customer needs. Because at least one pair programmer is continuously focused on the strategic direction of the coding process, there is an opportunity for the pair to remain cognizant of the client's needs as the coding progresses. Ancona (1990) described the probing (task coordinator) strategy as a process of interacting with outsiders to diagnose their needs and to experiment with solutions. If pair programmers continuously gather information on customer requirements, they can actively explore various solutions that are in line with those requirements. Timely user requirement information can be incorporated into the collaborative programming process in which partners can develop and experiment with solutions more effectively. Ultimately the solution developed by pair programmers will be a better reflection of customer needs when more feedback is provided.

Second, customer participation provides timely and detailed feedback to developers about the quality of the software. Customers may be better able to identify problems that developers are likely to overlook. Customers may be especially effective at identifying potential weaknesses in the current code, thus, enabling pair programmers to respond accordingly (Beck 2000). Nidumolu (1995) found a positive association between user participation (through horizontal coordination) and project performance. Hartwick and Barki (1994) also found that increased user involvement in systems development efforts yielded increased project performance. I argue that these boundary spanning activities enhance the benefits gained from pair programming, allowing the team to be more effective.

Hypothesis 1b: *Customer feedback activities will moderate the relationship between pair programming and software project team effectiveness such that higher levels of customer feedback will strengthen the relationship.*

Continuous Integration. Requirements in software projects are frequently subject to change. Hence, the software code must often be changed by adding, deleting, and/or adapting functionalities in order to keep pace with new business requirements. Making multiple changes to the software over time increases the likelihood that errors will occur in the code (Banker and Slaughter 2000). Further, there is a risk that modules of the code will become disjointed and cease to work optimally. As Littlewood and Strigini (1992) suggest, every little change or addition to existing software code can lead to a series of new errors. The opportunity for such errors to occur increases as the frequency with which changes to the software are made grows. As noted earlier, the current software development environment is clearly characterized by such frequent changes in requirements (Maurer and Martel 2002). Banker and Slaughter (2000) argue that software volatility—which they define as “the frequency or number of enhancements per unit of application functionality over a specified time frame” (p. 221)—increases the number of errors in software enhancement. Their empirical study of software applications in two different organizations yielded support for this argument.

The integration of changes or new additions to the base code occurs on a continual basis. One of the core features of agile approaches to programming is that newly created code is always tested to ensure that it runs at 100%. Unit tests are run on existing code until it runs flawlessly (Beck 2000). The actual process of integration involves a pair of programmers loading up the most recent release of the code, loading the newest set of changes, integrating the new changes with the current release, and running tests on the integrated code (Beck 1999, 2000). Often, one set of changes is run at a time so that if a test fails, the changes that caused the failure can be identified. Thus, whenever changes are made to existing code, pairs of programmers sit down and run unit tests on the newly integrated code. This ensures that a working baseline version of the project is always available on which all additions and modifications can be based (Maurer and Martel 2002). This process ensures that errors are uncovered and eliminated as changes are incorporated into the baseline code. Through continuous integration, the quality of the software product should improve as the likelihood of errors going unnoticed is reduced through frequent tests conducted by pairs of programmers. In addition to improved quality, continuous integration is expected to give the team a sense of confidence in its abilities, as a working version of the current release is always available. Indeed prior research suggests that time spent early on uncovering defects can improve effectiveness in terms of development effort and project quality (Jones 1997; Swanson, McComb, Smith and McCubbery 1991). Continuous integration is therefore expected to positively influence software project team effectiveness.

Hypothesis 2a: *Continuous integration will positively influence software project team effectiveness*.

Because continuous integration is an ongoing activity, developers are in a position to incorporate requirements changes provided by customers, even late in the software development process. The more feedback received the more that developers can integrate changes that meet customer requirements. Hence, higher levels of customer feedback enhance the effectiveness of continuous integration. Customers are in a prime position to provide feedback on requirements when they are involved in testing newly integrated code.

Hypothesis 2b: *Customer feedback will moderate the relationship between continuous integration and software project team effectiveness such that higher levels of customer feedback will strengthen the relationship.*

Hypothesis 2c: *Customer participation will moderate the relationship between continuous integration and software project team effectiveness such that higher levels of customer participation will strengthen the relationship.*

Refactoring. The refactoring practice in agile programming approaches involves the continuous effort of the software development team to simplify the software code while ensuring that the software continues to pass all tests (Beck 2000). Team members try to identify new and more efficient ways to make the software run without changing its functionality. Refactoring, thus, involves software-enhancing activities, which include the elimination of redundancies in the code and the reduction of complexity. For example, any time the system requires that code be duplicated, refactoring is needed (Beck 2000). Duplicate code increases the likelihood of new errors being introduced when changes to the code are made.

Software complexity is defined as a measure of the human effort required to comprehend, maintain, change, and test software (Basili 1980). Software code that is highly complex demands greater resources to maintain and change existing code (Banker, Davis and Slaughter 1998). Complexity in software code also increases the cognitive load on developers and maintainers (Curtis, Sheppard and Milliman 1979). Banker et al. (1998) conceptualize complexity in terms of three key dimensions: component, coordinative, and dynamic. Component complexity refers to the amount of information cues that need to be processed to perform a task. Coordinative complexity represents interdependencies between information cues. Finally, dynamic complexity is the result of changes in the nature of relationships between information cues (Banker et al., 1998). Software complexity has been found to negatively impact project performance (Banker, Datar, Kemerer and Zweig 1993).

Refactoring is expected to reduce complexity in software code in several ways. First, developers are constantly looking for ways to enable the code to process information in the simplest possible manner. This can be accomplished by eliminating redundancies in the software code (Beck 1999). Second, coordinative complexity can be managed by modularizing the code so that developers can comprehend, maintain, and test discrete components at a time. Modularization enables developers to focus on discrete interdependencies between components of the software (Banker et al. 1993; Conte Dunsmore and Shen 1986). Finally, refactoring reduces dynamic complexity through its emphasis on running unit tests on the code after changes have been made (Beck 2000). Further, simplifying the code through refactoring reduces the amount of effort required to integrate new software components later on, thus, enhancing the downstream productivity of the software project team. Hence, I expect refactoring to be positively associated with software project team effectiveness.

Hypothesis 3a: *Refactoring will positively influence software project team effectiveness.*

Refactoring is not only concerned with the simplification of existing code, but also with ensuring that the fundamental behavior of the software does not deviate from customer expectations (Beck 1999, 2000). In order to ensure that simplified software code continues to meet customer expectations, software development teams would be best served by soliciting customer participation in the testing of the software once it has been simplified. Customers offer the closest simulation of the actual environment in which the software will be used. Thus, they will be better able to assess the extent to which the software meets the needs for which it is developed. Further, contact with the customer during refactoring will allow developers to remain attuned to downstream expectations, thus, allowing them to better facilitate the integration of new software components later on. I, therefore, expect the benefits of refactoring to be enhanced when software project teams solicit the participation of their customers, thus, enabling the team to be more effective.

Hypothesis 3b: *Customer participation will moderate the relationship between refactoring and software project team effectiveness such that higher levels of customer participation will strengthen the relationship.*

Debate in Planning. Debate is defined as an open discussion of task-related differences among members of a social collective (Simons, Pelled and Smith 1999). It is a process that involves genuine dissent on task issues. Members of a team may have different approaches or viewpoints about how to solve problems (Schweiger, Sandberg and Rechner 1989). These approaches are voiced through the interactive process that is debate. Debate is expected to be an important aspect of software project teams' activities. Debate is most important during the planning stage, where the team determines the scope of the project, the design of the project, required resources etc. (Beck 2000). Software project teams that engage in debate are likely to discover a variety of solutions to a problem, compared to teams in which differing viewpoints are not voiced. Debate serves to debias the decision-making process, yielding better solutions to problems (Kray and Galinsky 2003; Schulz-Hardt, Jochims and Frey 2002). Thus, debate is expected to positively contribute to software project team effectiveness.

The effectiveness of solutions derived from active debate around problem solving is likely to be enriched and yield greater outcomes when customers are actively involved in the decision-making process. Prior research on systems development has shown that responsiveness to user feedback is an important determinant of the quality of the system (Hartwick and Barki 1994). Greater input from customers about requirements (during the debate) will ensure that the final solution is aligned with customer expectations (Kray and Galinsky 2003). Customers can provide important information during the planning stage, regarding how much functionality is sufficient to solve the business problem, which functionalities are the most critical, and what release dates are important for the client (Beck 2000). Such information enables the team to make more informed decisions and develop more informed potential solutions. I, therefore, expect team solicitation of customer participation in requirements determination to moderate the relationship between debate and software project team effectiveness.

Hypothesis 4a: *Debate in the planning process will positively influence software project team effectiveness.*

Hypothesis 4b: *Customer participation will moderate the relationship between debate and software project team effectiveness such that higher levels of customer participation will strengthen the relationship.*

Decision Comprehensiveness in Planning. The concept of decision comprehensiveness is distinct from debate. While both of these are process variables, they do have some underlying differences. Simons et al. (1999) describe decision comprehensiveness as “a process in which team members look at an issue with a wide lens, considering multiple approaches, multiple courses of action, and multiple criteria” (p. 663). In contrast to debate’s focus on voicing dissent, decision comprehensiveness emphasizes the weighing of a variety of alternatives, listing pros and cons, and offering constructive criticism of ideas (Simons et al. 1999). Further, it is possible for there to be debate within a team without decision comprehensiveness, and vice versa (Simons et al. 1999).

Decision comprehensiveness is important during the planning of the project design. The software project team is able to benefit from the diversity of expertise that is brought to bear by the developers on the team. Specifically, individual developers are able to provide their own ideas about the best way to proceed with the project and meet the client’s needs. By comprehensively identifying alternatives and examining the pros and cons of a variety of ideas the team is more likely to identify an effective approach to designing the software. By actively seeking and discussing alternative solutions and approaches regarding project design, software project teams ensure that they always have options in the event that a chosen alternative becomes problematic or if requirements should change in a way that makes the chosen alternative unviable. An additional benefit of decision comprehensiveness is that it enables team members to know where expertise is located within the team (Faraj and Sproull 2000). Hence, by exploring more alternatives, software project teams are able to remain adaptive to changing requirements, thus, improving their performance under uncertain conditions.

Finally, soliciting the involvement of the customer in decision comprehensiveness enables the software project team to verify the extent to which various alternatives will satisfy client requirements. Since the customer has a better idea of how the final software product will be used in the business environment, they can provide a more realistic assessment of the potential of a proposed solution. Hence, the customer can add value to the decision comprehensiveness process by actively weighing in on the identification and discussion of the pros and cons of proposed alternatives. Therefore:

Hypothesis 5a: *Decision comprehensiveness in planning will positively influence software project team effectiveness*.

Hypothesis 5b: *Customer participation will moderate the relationship between decision comprehensiveness and software project team effectiveness such that higher levels of customer participation will strengthen the relationship*.

Method

A longitudinal field study was conducted to test the proposed model. A few of the variables in the proposed model did not have existing scales. New scales were, therefore, developed prior to testing the model. In this section, I discuss the details of the scale development and pilot testing process. The participants, measurement, and data collection procedure are also discussed.

Pilot Test

Pair programming, continuous integration, refactoring, and unit testing are constructs that were new to this research. Hence, it was necessary to develop scales to measure these constructs. The details of the scale development process are outlined next.

Scale development. In order to capture the essence of a construct it is important to generate multi-item scales so as to reduce threats to content and construct validity and reliability that may arise when single-item measurement scales are used (Nunnally and

Bernstein 1994). The process of developing multi-item scales for constructs that are at a higher level of analysis—i.e., higher than the individual level—is somewhat complex. In addition to capturing information on the phenomenon of interest, it is also very important that measurement scales be specific to the level of the theory for which they are developed (Klein, Dansereau and Hall 1994). Theory, measurement, and statistical analysis need to be specified at the same level. Otherwise, there is a risk of producing erroneous results (Dansereau, Alutto and Yammarino 1984; Klein et al. 1994). Hence, team-level variables need to be measured at the level of the team.

The pair programming, refactoring, continuous integration and unit testing constructs are specified at the level of the team. Therefore, it is important that the measures also be specified at that level. However, in order to assess the extent to which software development teams use these programming practices, it is necessary to collect the data from individuals *within* the team. This issue is resolved by wording the scale items with respect to the *team* rather than the individual. Thus, the focus (or referent) of the scale items shifts from the individual (who is providing scale ratings) to the team, because the individual is providing responses about the team, not about himself or herself. This referent shift consensus addresses concerns about using individual-level responses to measure unit-level (in this case, team-level) variables (Chan 1998). As I have already noted, here, individual response items refer to *team-level* phenomena and is in keeping with the guidelines of Klein et al. (1994). To ensure that the measurement scales would capture team-level phenomena, all items used the team as the referent (Chan 1998; Kozlowski and Klein 2000).

Multi-item measurement scales were created to capture pair programming, refactoring, continuous integration, and unit testing practices in software project teams. Item pools were generated for each of the constructs. In order to ensure content validity, the items generated were based on conceptual definitions, descriptions, and narratives of the programming practices, as discussed in earlier sections and elaborated in practitioner websites such as AgileAlliance.[1] Once items had been generated for each construct, a developer who was familiar with agile methodologies evaluated the pool of items, made suggestions for refinement of some items, and suggested a few others. Straub (1989) suggests that such evaluation and refinement helps improve the content validity of the scale. Based on the suggestions, additions, deletions, and modifications were made.

The final item pool for pair programming consisted of eight items. Sample items included: "Most of the programming on this team is done by pairs of developers," "Software developers on this team generally code software on their own, *without* a partner," and "On this team, we do our software development using pairs of programmers." The item pool for continuous integration included eight items e.g., "We combine new code with existing code on a continual basis," "How often do members of this team combine newly coded units of software with the current software release?" and "Each time a new unit is coded, members of this team integrate the software code." To develop the measurement scale for refactoring, seven items were generated. Sample items for refactoring included: "Where necessary, members of this team try to simplify existing code without changing its functionality," "We periodically identify and eliminate redundancies in the software code," and "As long as the software performs as it should, members of this team *are not* concerned with simplifying the code." Finally, five items were generated to capture unit testing e.g., “On this team, coding additional functions does not proceed until existing code runs flawlessly,” “Members of this team actively engage in unit testing,” “We code the next set of functions on a project *without testing* existing code.” A complete list of the items generated for the programming practices can be found in Appendix 2.A.

A group of 10 doctoral students was asked to participate in a card sorting exercise involving the items generated to measure the constructs of interest. Each card in the sorting exercise had a single item on it. All of the items were placed (in randomized arrangement) in a source pile. Participants were asked to sort the items in the source pile into bins of items they perceived to be related. Participants were allowed to create as many item bins as they felt was appropriate. The card sorting exercise was part of a wider scale development exercise involving additional programming practices that were outside the scope of the current paper. The card sorting exercise yielded an interrater agreement of 0.84 for the items pertaining to the constructs of interest—i.e., pair programming, refactoring, continuous integration, and unit testing. Given the level of agreement among participants in the card sorting exercise, we decided to conduct a pilot test that included all of the items generated for the four constructs.

Participants. Participants in the field study were software developers, analysts, and project managers. In all, 1,377 potential participants were contacted. A total of 149 usable responses were received for an effective response rate of 11%. All participants were actively engaged in ongoing software development projects. Of the 149 participants, 19 were women (13%). The average age was 35 with a standard deviation of 8.06. Finally, the participants had an average of 13 years of experience as programmers (with a standard deviation of 7.79).

Procedure. An email was sent to potential participants, describing the purpose of the study and providing a link to the survey. Participants were told that the purpose of the study was to understand the use of programming practices in software development teams. Because the participants were engaged in ongoing software development projects, they were instructed to complete the survey with respect to the current project. In cases where participants were involved in more than one project, they were instructed to choose one and respond accordingly. One month after the survey was sent out, 913 reminders were issued to participants. A mean comparison was conducted to ensure that late response bias was not a concern. No mean differences were detected.

Analysis. Inter-item correlations and factor analysis results were used to arrive at a final set of items for each construct. The means, standard deviations, and inter-item correlations are presented in Table 2.1. As Table 2.1 illustrates, many of the within-construct inter-item correlations were higher than the across-construct inter-item correlations. However, there were several items that had high cross-construct correlations. To further examine the relationships among the items, a principal components factor analysis was conducted. Although the final sample size was small, recent research (Kirsch et al. 2002) and guidelines (Stevens 1996; Tinsley and Tinsley 1987) suggest that a ratio of five observations per item is acceptable for factor analysis.

Table 2.1: Inter-item Correlations for Pilot Test.

Items	Mean	SD	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1. PP1	2.44	1.35	1															
2. PP2	2.41	1.26	**.85*****	1														
3. PP3	2.48	1.36	**.90*****	**.84*****	1													
4. PP4	2.35	1.40	**.90*****	**.84*****	**.95*****	1												
5. PP5	2.17	1.37	**.89*****	**.84*****	**.87*****	**.90*****	1											
6. PP6	2.50	1.34	**.87*****	**.83*****	**.83*****	**.86*****	**.88*****	1										
7. PP7	2.54	1.34	**.88*****	**.81*****	**.85*****	**.86*****	**.88*****	**.97*****	1									
8. PP8	2.11	1.17	**.75*****	**.83*****	**.71*****	**.74*****	**.77*****	**.77*****	**.76*****	1								
9. CI1	3.83	1.23	.39***	.31***	.35***	.34***	.42***	.47***	.45***	.34***	1							
10. CI2	4.17	.96	.27***	.17*	.25**	.24**	.34***	.35***	.32***	.20*	**.64*****	1						
11. CI3	4.12	.98	.27***	.17*	.25**	.25**	.28**	.36***	.34***	.18*	**.57*****	**.67*****	1					
12. CI4	4.12	.93	.13	.22**	.18*	.20*	.22**	.25**	.23**	.23**	**.35*****	**.56*****	**.45*****	1				
13. CI5	4.15	.85	.20*	.27**	.27**	.25**	.31***	.30***	.27**	.31***	**.49*****	**.55*****	**.48*****	**.50*****	1			
14. CI6	3.97	.96	.28**	.35***	.26**	.26**	.36***	.36***	.34***	.40***	**.61*****	**.63*****	**.55*****	**.56*****	**.72*****	1		
15. CI7	4.14	.92	.37***	.25**	.38***	.36***	.43***	.47***	.45***	.28**	**.66*****	**.72*****	**.74*****	**.45*****	**.57*****	**.62*****	1	
16. CI8	4.14	.94	.35***	.21*	.33***	.32***	.39***	.43***	.42***	.25**	**.68*****	**.69*****	**.75*****	**.40*****	**.56*****	**.64*****	**.87*****	1
17. REF1	3.73	1.08	.34***	.29**	.34***	.32***	.38***	.36***	.34***	.26**	.50***	.47***	.46***	.24**	.38***	.43***	.55***	.56***
18. REF2	3.62	1.00	.37***	.32***	.44***	.46***	.44***	.43***	.44***	.31***	.41***	.37***	.42***	.28**	.27**	.37***	.48***	.40***
19. REF3	3.91	1.06	.11	.16†	.08	.09	.20*	.16†	.18*	.19*	.41***	.35***	.26**	.31***	.33***	.47***	.35***	.38***
20. REF4	3.59	1.10	.36***	.36***	.27**	.30***	.38***	.35***	.36***	.39***	.32***	.31***	.26**	.17*	.35***	.45***	.36***	.39***
21. REF5	3.32	1.06	-.17*	-.26**	-.16†	-.19*	-.23**	-.22**	-.20*	-.29**	-.15†	.03	.10	.03	-.08	-.10	.08	.08
22. REF6	3.55	.90	-.05	-.18*	-.09	-.13	-.14†	-.12	-.11	-.21*	-.04	.13	.21*	-.06	-.03	.01	.14†	.19*
23. REF7	3.55	.98	-.11	-.09	-.15†	-.19*	-.13	-.15†	-.14†	-.06	-.04	-.01	.08	-.02	-.05	.02	.03	.06
24. UT1	3.91	1.17	.32***	.29**	.42***	.42***	.38***	.39***	.37***	.24**	.42***	.50***	.49	.40***	.38***	.37***	.51***	.44***
25. UT2	4.01	1.12	.32***	.25***	.34***	.35***	.37***	.39***	.38***	.24**	.53***	.58***	.59***	.44***	.38***	.51***	.62***	.62***
26. UT3	3.37	1.26	.39***	.34***	.34***	.37***	.37***	.45***	.44***	.34***	.39***	.40***	.34***	.23**	.28**	.42***	.50***	.46***
27. UT4	3.98	1.05	.32***	.35***	.28***	.27***	.27***	.36***	.35***	.34***	.43***	.35***	.31***	.30***	.35***	.58***	.43***	.42***
28. UT5	4.10	1.04	.37***	.32***	.37***	.38***	.38***	.43***	.41***	.26**	.59***	.53***	.49***	.36***	.25**	.44***	.57***	.62***

(*continued on next page*)

Notes:
1. PP = pair programming; CI = continuous integration; REF = refactoring; UT = unit testing. Within construct inter-item correlations bolded.
2. n = 149.
3. † p < .10; * p < .05; ** p < .01; *** p < .001.

Table 2.1 (continued).

Items	17	18	19	20	21	22	23	24	25	26	27
17. REF1	1										
18. REF2	**.68*****	1									
19. REF3	**.58*****	**.49*****	1								
20. REF4	**.71*****	**.52*****	**.64*****	1							
21. REF5	**.16**†	**.15**†	**.18***	**.04**	1						
22. REF6	**.25****	**.16**†	**.13**	**.09**	**.77*****	1					
23. REF7	**.22****	**.10**	**.26****	**.20***	**.39*****	**.34*****	1				
24. UT1	.44***	.48***	.29**	.26**	.04	.05	-.04	1			
25. UT2	.48***	.40***	.34***	.32***	.02	.05	.03	**.77*****	1		
26. UT3	.43***	.45***	.28**	.33***	-.07	.01	-.04	**.47*****	**.46*****	1	
27. UT4	.50***	.45***	.46***	.49***	-.05	-.01	.13	**.39*****	**.46*****	**.62*****	1
28. UT5	.48***	.38***	.34***	.33***	-.06	.02	.03	**.68*****	**.82*****	**.47*****	**.47*****

Notes:
1. REF = refactoring; UT = unit testing. Within construct inter-item correlations are in bold font.
2. $n = 149$.
3. † $p < .10$; * $p < .05$; ** $p < .01$; *** $p < .001$.

Table 2.2: Factor Analysis with Oblimin Rotation for Pilot Test.

	Factors				
Items	1	2	3	4	5
PP1	**.960**	.024	.019	.066	.031
PP2	**.905**	.005	.115	.112	.100
PP3	**.945**	.020	.086	.056	.084
PP4	**.948**	.043	.069	.011	.103
PP5	**.887**	.042	.054	.049	.070
PP6	**.888**	.105	.023	.022	.074
PP7	**.898**	.071	.012	.001	.072
PP8	**.811**	.097	.175	.162	.186
CI1	.115	**.550**	.112	.091	.244
CI2	.004	**.738**	.035	.085	.229
CI3	.056	**.673**	.127	.254	.263
CI4	.023	**.680**	.001	.070	.028
CI5	.060	**.852**	.127	.100	.218
CI6	.028	**.774**	.301	.147	.064
CI7	.140	**.661**	.013	.179	.275
CI8	.096	**.666**	.019	.193	.259
REF1	.132	.130	**.585**	.231	.243
REF2	.268	.008	**.450**	.203	.285
REF3	.106	.181	**.770**	.052	.032
REF4	.185	.067	**.789**	.039	.047
REF5	.081	.021	.014	**.882**	.023
REF6	.012	.052	.030	**.903**	.023
REF7	.110	.087	.412	**.473**	.116
UT1	.079	.070	.002	.014	**.780**
UT2	.008	.216	.033	.004	**.778**
UT3	.153	.002	.292	.121	**.519**
UT4	.032	.069	**.608**	.187	.324
UT5	.050	.081	.079	.053	**.806**

Notes:
1. $\delta = -0.20$.
2. $R^2 = 0.82$.

Because the various agile programming practices are related, items were expected to correlate across constructs (i.e., factor solutions were not expected to be orthogonal). Thus, direct oblimin rotation was used. Direct oblimin rotation allows the researcher to specify the degree to which factors can be correlated by varying the value of δ (Harman 1976). The procedure obtains a factor pattern without involving an intermediate reference structure. Harman (1976) recommends negative or zero δ values to obtain oblique factors. Thus, a δ of -0.20 was used in the factor analysis. The results of the factor analysis are presented in Table 2.2.

Based on an examination of the inter-item correlations and the factor structure, a final set of items was selected. The items for pair programming had the highest correlations. PP1, PP3, PP4, PP5, and PP7 were selected because of their high inter-item correlations, which ranged from 0.85 to 0.95, and their high loadings, which ranged from 0.89 to 0.96. For continuous integration, CI2, CI3, CI6, CI7, and CI8 were selected because they had high correlations ranging from 0.67 to 0.87. These items also had high loadings on the continuous integration factor, ranging from 0.66 to 0.77. Two factors emerged from the refactoring items. The loadings on the third factor (REF1, REF2, REF3, and REF4) ranged from 0.45 to 0.79 and the loadings on the fourth factor (REF5, REF6, and REF7) ranged from 0.47 to 0.90. Aside from the correlation between REF5 and REF6 of 0.77, the correlations between items on the fourth factor were less than 0.40). The wording for these items seemed to reflect the extent to which developers try to maintain flexibility in their code. The items for the third factor pertained to simplifying code, which is closer to the concept of refactoring. Thus, the items comprising the third factor were retained. Specifically, REF1, REF3, and REF4 were retained and REF2 was dropped. The correlations between REF1, REF3, and REF4 ranged from 0.64 to 0.71. REF2 only correlated highly with REF1 ($r = 0.68$). Further, REF2 had the lowest loading (0.45) of the items forming the refactoring factor. Finally, UT1, UT2, and UT5 were selected for the unit testing scale. One item, UT4, loaded on a different factor and was, therefore, dropped. The item pertained to software project teams' coding of software functions without unit testing. UT3 had the lowest loading on the unit testing factor (0.52). The loadings for the three selected items ranged from 0.78 to 0.81. Further, these items had the highest correlations among the unit testing items, ranging from 0.68 to 0.82. The inter-item correlations for the selected set of items are presented in Table 2.3. The selected items were once again factor analyzed using principal components analysis with direct oblimin rotation. As Table 2.4 illustrates, the resulting factor structure had an R^2 of 0.83.

Reliabilities were calculated for the selected sets of items. The reliabilities of the final scales exceeded 0.70, suggesting adequate reliability. The reliabilities were as follows: pair programming α = 0.98, continuous integration α = 0.91, refactoring α = 0.85, and unit testing α = 0.90. In order to assess convergent and discriminant validity, inter-construct correlations were calculated. In addition, the average variance extracted was calculated for each construct. The construct correlations are displayed in Table 2.5 with the square root of the average variance extracted on the diagonal. As the results in Table 2.5 show, the square root of the average variance extracted was larger than the inter-construct correlations for all four constructs, indicating adequate convergent and discriminant validity for the new scales. The final set of scales is shown in Table 2.6. One of the items for refactoring (REF4) was reworded and two additional items were added to the scale. The newly validated scales were field tested as part of the overall theoretical model testing. The details of the field study are discussed next.

Table 2.3: Inter-item Correlations for Selected Items.

Items	Mean	SD	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1. PP1	2.44	1.35	1														
2. PP3	2.48	1.36	**.90*****	1													
3. PP4	2.35	1.40	**.90*****	**.95*****	1												
4. PP5	2.17	1.37	**.89*****	**.87*****	**.90*****	1											
5. PP7	2.54	1.34	**.88*****	**.85*****	**.86*****	**.88*****	1										
6. CI2	4.17	.96	.27**	.25**	.24**	.34***	.32***	1									
7. CI3	4.12	.98	.27**	.25**	.25**	.29***	.34***	**.67*****	1								
8. CI6	3.97	.96	.28**	.26**	.26**	.36***	.34***	**.63*****	**.55*****	1							
9. CI7	4.14	.92	.37***	.38***	.36***	.43***	.45***	**.72*****	**.74*****	**.63*****	1						
10. CI8	4.14	.94	.35***	.33***	.32***	.39***	.42***	**.69*****	**.75*****	**.64*****	**.87*****	1					
11. REF1	3.73	1.08	.34***	.34***	.32***	.38***	.34***	.47***	.46***	.43***	.55***	.56***	1				
12. REF3	3.91	1.06	.11	.08	.09	.20*	.18*	.35***	.26**	.47***	.35***	.38***	**.58*****	1			
13. REF4	3.59	1.10	.36***	.27**	.30***	.38***	.36***	.31***	.26**	.45***	.36***	.39***	**.71*****	**.64*****	1		
14. UT1	3.91	1.17	.32***	.42***	.42***	.38***	.37***	.50***	.49***	.37***	.51***	.44***	.44***	.29**	.26**	1	
15. UT2	4.01	1.12	.32***	.34***	.35***	.37***	.38***	.58***	.59***	.51***	.62***	.62***	.48***	.34***	.32***	**.77*****	1
16. UT5	4.10	1.04	.37***	.37***	.38***	.42***	.41***	.53***	.49***	.44***	.57***	.62***	.48***	.34***	.33***	**.68*****	**.82*****

Notes:
1. PP = pair programming; CI = continuous integration; REF = refactoring; UT = unit testing. Within construct inter-item correlations are in bold font.
2. *n* = 149.
3. † $p < .10$; * $p < .05$; ** $p < .01$; *** $p < .001$.

Table 2.4: Factor Analysis with Direct Oblimin Rotation (Selected Items) for Pilot Test.

	Factors			
Items	1	2	3	4
PP1	**.948**	.034	.035	.027
PP3	**.942**	.020	.035	.089
PP4	**.953**	.048	.016	.098
PP5	**.904**	.062	.082	.020
PP7	**.887**	.125	.027	.000
CI2	.016	**.783**	.019	.125
CI3	.007	**.839**	.092	.121
CI6	.045	**.710**	.249	.084
CI7	.116	**.825**	.017	.084
CI8	.068	**.831**	.068	.066
REF1	.087	.151	**.684**	.171
REF3	.133	.036	**.861**	.082
REF4	.159	.014	**.892**	.047
UT1	.081	.025	.026	**.886**
UT2	.004	.198	.046	**.809**
UT5	.071	.101	.086	**.787**

Notes:
1. $\delta = -0.20$.
2. $R^2 = 0.83$.

Table 2.5: Construct Correlations for Pilot Test.

Variable	Mean	S.D.	1	2	3	4
1. Pair Programming	2.40	1.34	**.96**			
2. Continuous Integration	4.06	.90	.42***	**.88**		
3. Refactoring	3.68	1.00	.36***	.57***	**.89**	
4. Unit Testing	3.95	1.09	.46***	.71***	.50***	**.92**

Notes:
1. *n* = 149.
2. Square root of average variance extracted on diagonal.
3. All correlations significant at p < .001.

Table 2.6: Final Scale Items.

Pair Programming	
PP1	How often is pair programming used on this team?
PP2	On this team, we do our software development using pairs of developers
PP3	To what extent is programming carried out by pairs of developers on this team?
PP4	We do pair programming a lot on this team
PP5	Software developers on this team generally code software on their own, without a partner (r)
Continuous Integration	
CI1	Members of this team integrate newly coded units of software with existing code
CI2	We combine new code with existing code on a continual basis
CI3	We do not spend time integrating new units of code with existing code (r)
CI4	Combining recently coded units of the project with other units of the code is not carried out by members of this team (r)
CI5	Our team does not take time to combine various units of code as they are developed (r)
Refactoring	
REF1	Where necessary, members of this team try to simplify existing code without changing its functionality
REF2	We periodically identify and eliminate redundancies in the software code
REF3	We do not eliminate redundancies in the software code (r)
Added	We continuously try to reduce complexity in the software code
Added	We periodically simplify existing code
Unit Testing	
UT1	We run unit tests on newly coded modules until they run flawlessly
UT2	Members of this team actively engage in unit testing
UT3	To what extent are unit tests run by this team?

Model Testing

A longitudinal field study was conducted to test the research model and hypotheses. This field setting enhanced the external validity of the research. The study spanned four months and involved three points of measurement. The setting, participants, measurement, and data collection process are described in this section.

Participants. Participants were employees in a software development firm. The organization was the site of a number of newly initiated software development projects. The focus of this study was the employees (developers, analysts etc.) who were members of the teams that were engaged in the software development projects. Out of the 79 teams that were involved in software projects, 73 agreed to participate. Of these 73 teams, 56 provided usable responses at all three points of measurement in the study for an effective response rate of 77%. While ideally I would have wanted all teams to participate in the entire study, this was not feasible given that the duration of the study was four months and involved 3 points of measurement. Of the 509 participants 28% were women. The average age of participants was 29.4 with a standard deviation of 4.90. The participants had an average of 7.22 years of programming experience with a standard deviation of 2.54. Finally, the average team size was 10 (range: 8 to 11).

Measurement

Table 2.7 presents the measures that were used in the research study. Where possible, variables in the model were operationalized by adapting existing scales. The newly developed scales for pair programming, continuous integration, refactoring, and unit testing were also used to test the research model. As justification for aggregation of individual level responses to team level scores, the average r_{wg} for each team level variable is reported. The r_{wg} assesses the extent to which raters' (team members) responses to team-level items are convergent (James, Demaree and Wolf 1984). An r_{wg} value of .70 or higher indicates adequate interrater agreement.[2] The intraclass correlation I (ICC[1]), which assesses the amount of variability in the team-level measures, is also reported for each construct (Bliese 2000). ICC(1)[3] values between .05 and .25 are common in organizational research.

Table 2.7: Measures Used in the Research Study.

CONSTRUCT	SOURCE
Task-focused Activities:	
Pair programming	Scale developed
Continuous integration	Scale developed
Refactoring	Scale developed
Decision Making Activities:	
Debate in planning	Adapted from Simons, Pelled, and Smith (1999)
Decision comprehensiveness in planning	Adapted from Simons, Pelled, and Smith (1999)
Boundary Spanning Activities:	
Customer feedback	Adapted from Ancona and Caldwell (1992)
Customer participation	Adapted from Ancona and Caldwell (1992)
Team Effectiveness:	
Team-rated effectiveness	Adapted from Nidumolu (1995)
Leader-rated effectiveness	Adapted from Nidumolu and Subramani (2004)

Decision-making processes. Debate in planning was measured using a four-item scale developed by Simons et al. (1999). The scale assesses the extent to which team members challenge one another's opinions during the planning stage. Sample items from this scale include: "In discussions about the project, team members state clear disagreements with each other" and "team members openly challenge each other's opinions." The Cronbach alpha for this scale was .75. A 3-item scale for decision comprehensiveness was also adapted from Simons et al. (1999). The scale captures the extent to which the team is thorough in its exploration of alternative solutions during the planning process. A sample item is "During planning, to what extent does your team examine the pros and cons of several possible courses of action?" The decision comprehensiveness scale had a reliability of .78. The average r_{wg} for debate and decision comprehensiveness was .77and .74 respectively. The ICC(1) for each scale was .21 and .28 respectively.

Task-focused processes. As mentioned earlier, the newly developed scales were used to measure the teams' task-focused activities. The 5-item scale was used to measure pair programming. The scale had a Cronbach alpha of .83. The scale to measure continuous integration included five items, which had a reliability of .82. Finally, a 5-item scale was used to measure refactoring, which had a reliability of .88. A list of the items used to measure team task-focused activities can be found in Table 2.7. The newly developed scales demonstrated good team-level properties. The average r_{wg} for pair programming, continuous integration, and refactoring was .67, .78, and .75 respectively. The ICC(1) for pair programming, continuous integration, and refactoring was .15, .22, and .24 respectively. Finally, the three scales each had an ICC(2)[4] value greater than .70 suggesting adequate reliability at the team level (pair programming ICC(2) = .75; continuous integration ICC(2) = .80; refactoring ICC(2) = .76).

Boundary spanning activities. A 4-item scale was adapted from Ancona and Caldwell (1992) to measure customer feedback activities. The scale assesses the extent to which the team communicates with the customer. A sample item for this scale is "Our team negotiates delivery deadlines and schedules with the customer." The reliability for this scale was .80. Customer participation was measured using six items also adapted from Ancona and Caldwell (1992). The scale captures the extent to which the team actively involves its customer in team activities. Sample items for this scale include "We make an effort to involve the client in software development," "Members of our team encourage the client to participate in testing code," and "On this team, we ask the client to help us to make decisions about the project design." The scale had a reliability of .77.

The average r_{wg} for task coordinator activity and soliciting participation was .74 and .79 respectively and the ICC(1) values were .29 and .14 respectively.

Team effectiveness. Objective performance measures can be problematic in IS research for a variety of reasons (Faraj and Sproull 2000; Henderson and Lee 1992). Metrics such as lines of code are often difficult to obtain. Even when objective measures are available, it is inappropriate to make cross project comparisons because software projects may differ on a number of dimensions. Thus, I operationalized team effectiveness by asking team project leaders to provide ratings of team effectiveness. The use of expert judgment as a measure of software project team effectiveness is acceptable in IS research (see Faraj and Sproull 2000; Henderson and Lee 1992 for examples). Team effectiveness was measured from team leaders, using a 4-item scale that included such items as "I am satisfied with this team's work on this project so far," and "The work that this team has done so far is of a high quality." The reliability of this scale was .80. I also asked team members to provide self-ratings of the quality of their work on the project. Team effectiveness was measured from team members using a 4-item scale. The scale included such items as "Development and acceptance tests indicate that our code contains numerous defects/errors." The scale had a reliability of .84. The average r_{wg} for this scale was .75 and the ICC(1) value was .19. A detailed list of the scales adapted for this study can be found in Appendix 2.B.

Control variables. Prior research has found team size and average team member experience to be important predictors of team performance. These were therefore included as controls. The number of years of programming experience for members within each team was averaged to create a team score of experience. Testing of software has been found to be important for software project performance. Thus, unit testing was used as a control variable. The unit testing scale had a Cronbach α of .85, an average r_{wg} value of .79, and an ICC(1) value of .19. Finally, unit testing had an ICC(2) of .72.

Procedure

Data were collected toward the beginning, during the middle, and towards the end of the software projects in naturally occurring conditions within the organization. The organization launched a new large-scale project, which was expected to last four months. The project required an integrated solution to be built for a client organization. The client organization's existing system had multiple fragmented systems across various business functions. The project involved 79 software development teams. The project teams were assembled for the purpose of the project. Each of the seventy-nine teams was to build a specific module for the software solution. The project teams each had a team leader whose responsibility was to manage the project through to completion. The team leaders held weekly meetings with their respective teams for status updates and to discuss major project-related challenges. Team leaders had standardized evaluation criteria on which to rate team effectiveness. These criteria were tied to how well the project met client requirements. Each team had its own (non-overlapping) client member with whom to interact. However, the teams had discretion over whether, and to what extent, they interacted with their assigned client member. Specifically, the client member of each team could attend weekly meetings or communicate with the team via telephone or email. All seventy-nine project teams began working on the projects simultaneously.

Participants were informed that the purpose of the study was to examine software project team processes. In order to organize team responses, all team members and team leaders were instructed to agree on a single team name, which would be used to track the questionnaires. Further, participants were instructed to provide their individual names and team name so that responses could be tracked over time. Participants were assured of the confidentiality of their responses and that only aggregate data would be reported.

One week after the official launch of the projects, participants filled out the first questionnaire. Five weeks later, when the teams were well into their software projects, the second questionnaire was administered. Team members responded to questions about their team's task-focused and decision-making processes. 11 weeks later, the third and final questionnaire, pertaining to team effectiveness, was administered. Data on team effectiveness were collected from team leaders and team members. Figure 2.3 presents the study design including all points of measurement.

Analysis

The convergent and discriminant validity of the constructs in the model was evaluated using factor analysis. As evidence of convergent validity, all items loaded, as expected, on the pre-specified constructs (see Table 2.8). The construct correlations are displayed in Table 2.9. The research model was tested using regression analysis. Control variables were entered into the regression model first, followed by the team process variables. The interaction terms were entered in the final step. Following guidelines outlined by Aiken and West (1991), the task-focused, decision-making, and boundary-spanning variables were standardized before creating the interaction terms. Such standardization reduces collinearity between the main effects and interaction terms. In order to understand the form of the moderation, the significant interactions were plotted at plus and minus one standard deviation from the mean (Aiken and West 1991).

Table 2.8: Factor Analysis with Direct Oblimin Rotation.

	Factors									
Items	1	2	3	4	5	6	7	8	9	10
LEFFECT1	**.72**									
LEFFECT2	**.74**									
LEFFECT3	**.70**									
LEFFECT4	**.71**									
TEFFECT1		**.79**								
TEFFECT2		**.71**								
TEFFECT3		**.74**								
TEFFECT4		**.71**								
PP1			**.82**							
PP2			**.90**							
PP3			**.77**							
PP4			**.85**							
PP5			**.65**							
CI1				**.72**						
CI2				**.81**						
CI3				**.65**						
CI4				**.83**						
CI5				**.88**						
REF1					**.85**					
REF2					**.80**					
REF3					**.77**					
REF4					**.80**					
REF5					**.66**					
UT1						**.80**				
UT2						**.71**				
UT3						**.70**				
DCOMP1							**.71**			
DCOMP2							**.78**			
DCOMP3							**.70**			
DEBATE1								**.77**		
DEBATE2								**.68**		
DEBATE3								**.72**		
DEBATE4								**.70**		
CFEED1									**.74**	
CFEED2									**.71**	
CFEED3									**.80**	
CFEED4									**.73**	
CPART1										**.67**
CPART2										**.62**
CPART3										**.70**
CPART4										**.71**
CPART5										**.69**
CPART6										**.74**

Notes:

1. LEFFECT = leader-rated effectiveness; TEFFECT = team-rated effectiveness; PP = pair programming; CI = continuous integration; REF = refactoring; UT = unit testing; CFEED = customer feedback; CPART = customer participation.
2. Loadings less than .25 not shown.
3. $\delta = -0.20$

Results

Factor Analysis

A factor analysis of all items, including those for the newly developed agile practices, was conducted using direct oblimin rotation. Consistent with the factor analysis in the pilot testing, a δ of -.20 was used. The results of the factor analysis are presented in Table 2.8. As the results indicate, all items loaded on the expected factors, with all cross-loadings being lower than .25. The high within-construct loadings and low cross-loadings suggest adequate convergent and discriminant validity for the scales. Further, the square root of the average variance extracted (AVE)[5] for each scale was greater than the inter-construct correlations, thus, establishing divergent validity for the scales.

Main Effects Hypotheses

Table 2.9 presents the correlations, means, and standard deviations for the constructs measured in the study. The results of the regression analysis predicting leader-rated and team-rated team effectiveness are presented in Table 2.10. A main effects model was used to test hypotheses 1a, 2a, 3a, 4a, and 5a. The moderation hypotheses, 1b, 2b, 2c, 3b, 4b, and 5b were tested by entering the interaction terms into the regression model after controlling for the main effects.

Model 2 of Table 2.10 presents the main effects regression results. The results for the models predicting leader-rated team effectiveness are reported first. The main effects

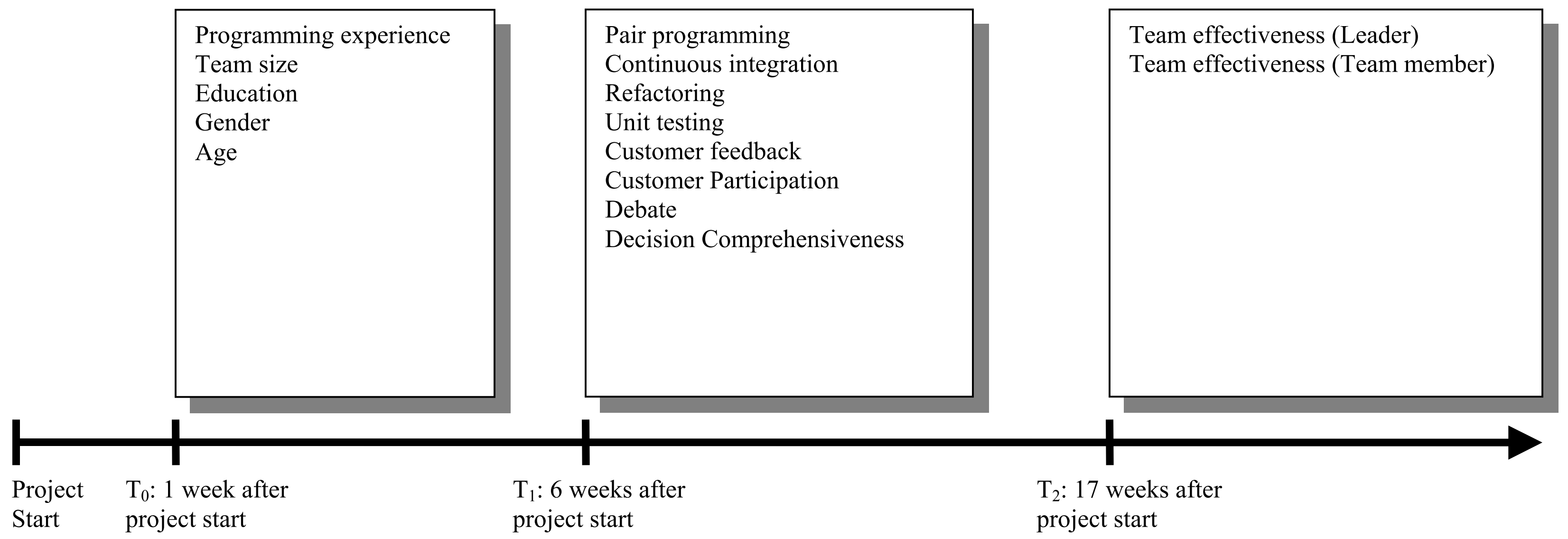

Programming experience
Team size
Education
Gender
Age
Pair programming
Continuous integration
Refactoring
Unit testing
Customer feedback
Customer Participation
Debate
Decision Comprehensiveness
Team effectiveness (Leader)
Team effectiveness (Team member)
Project Start
T_0: 1 week after project start
T_1: 6 weeks after project start
T_2: 17 weeks after project start

Figure 2.3: Summary of Study Design with Points of Measurement.

model (model 2) yielded an R^2 of .193 ($F_{10, 55} = 2.314$, $p < .05$).

The beta coefficient for pair programming was positive and significant ($\beta = .232$, $p < .01$), yielding support for hypothesis 1a. Hypothesis 2a predicted that continuous integration would positively influence team effectiveness. As the results indicate, the coefficient for continuous integration was positive and significant ($\beta = .430$, $p < .01$), supporting hypothesis 2a. Refactoring had a positive and significant beta coefficient ($\beta = .152$, $p < .05$), supporting hypothesis 3a. The coefficient for debate was non-significant ($\beta = -.048$, p = ns). Therefore, hypothesis 4a was not supported. Finally, Hypothesis 5a stated that decision comprehensiveness in the planning process would positively influence team effectiveness. The coefficient for decision comprehensiveness was positive and significant ($\beta = .434$, $p < .001$). Thus, hypothesis 5a was supported.

The main effects model (model 2) predicting team member-rated team effectiveness had an R^2 of .118 ($F_{10, 55} = 1.736$, $p = .10$). The pattern of results was the same as those for leader-rated team effectiveness. The coefficients for pair programming ($\beta = .114$, $p < .05$), continuous integration ($\beta = .232$, $p < .05$), refactoring ($\beta = .110$, $p < .05$), and decision comprehensiveness ($\beta = .257$, $p < .01$) were all positive and significant. Thus, hypotheses 1a, 2a, 3a, and 5a were supported. The coefficient for debate was non-significant ($\beta = -.026$, p = ns). Hypothesis 4a was therefore not supported.

Moderation Hypotheses

As noted earlier, the moderation hypotheses were tested by entering the interaction terms into the regression model after having entered the control variables and main effects. The results of the model predicting leader-rated team effectiveness are reported first. The interaction model (model 3) explained 48.3% of the variance in

Table 2.9: Correlations, Means, and Standard Deviations.

Variable	Mean	S.D.	ICR	1	2	3	4	5	6	7	8	9	10	11	12
1. EffectLeader	5.21	1.47	.72	**.80**											
2. EffectTeam	4.99	1.44	.74	.93***	**.84**										
3. PP	4.44	1.30	.80	.06	-.03	**.83**									
4. CI	4.81	1.11	.78	.05	-.05	.03	**.82**								
5. REF	4.17	1.22	.78	-.06	.02	-.10	-.23†	**.88**							
6. DCOMP	4.07	1.08	.73	.23†	.22	-.59***	-.30*	-.24†	**.78**						
7. Debate	4.33	1.14	.72	.06	.11	.10	-.06	-.06	.03	**.75**					
8. CFEED	4.01	1.32	.75	.17	.26†	-.11	-.44**	.16	.04	.12	**.80**				
9. CPART	4.43	1.17	.69	.05	.12	.30*	-.05	-.23†	-.25†	.47***	.02	**.77**			
10. UT	4.43	1.09	.74	.04	.11	-.30*	.06	.08	.10	-.00	.18	-.16	**.85**		
11. Team Size	10	1.33	-	.09	.06	.10	-.19	-.02	-.10	-.20	.14	-.00	.02	-	
12. PROGEXP	7.22	2.54	-	-.00	.03	.11	-.11	.03	-.06	-.13	-.08	.11	.14	.17	-

Notes:
1. ICR = Internal consistency reliability; EffectLeader = Team effectiveness (leader-rated); EffectTeam = Team effectiveness (team member-rated); PP = pair programming; CI = continuous integration; REF = refactoring; DCOMP = decision comprehensiveness; CFEED = customer feedback; CPART = customer participation; UT = unit testing; PROGEXP = average programming experience. Reliabilities (Cronbach-α) are provided on the diagonal.
2. n = 56.
3. † $p < .10$; * $p < .05$; ** $p < .01$; *** $p < .001$.

Table 2.10: Regression Analysis Predicting Leader-rated and Team-rated Effectiveness.

	Team effectiveness (project leader-rated)			Team effectiveness (team member-rated)		
	Model 1	Model 2	Model 3	Model 1	Model 2	Model 3
Controls:						
1. UT	.018	.017	.106	.037	.029	.085†
2. Team size	.044	.078	.075	.019	.037	.039
3. Programming Experience	-.011	-.005	.063	.001	.004	.051
Main Effects:						
4. PP		.232**	.888**		.114*	.554*
5. CI		.430**	-.219		.232*	-.283
6. REF		.152*	.025		.110*	.042
7. DCOMP		.434***	.379**		.257**	.228*
8. DEBATE		-.048	.211*		-.026	.150†
9. CFEED		.188*	.275		.134*	.158
10. CPART		.081	.234		.073†	.207
Interaction Terms:						
11. CFEED x PP			-.087			-.059
12. CFEED x CI			.244*			.184*
13. CPART x CI			.024			.011
14. CPART x REF			.127**			.077*
15. CPART x DEBATE			-.188**			-.126**
16. CPART x DCOMP			.218**			.142**
Model F	.172	2.314*	4.213***	.280	1.736	2.670**
df	52	45	39	52	45	39
R^2	.010	.340	.634	.016	.278	.523
Adjusted R^2	-.047	.193	.483	-.041	.118	.327
ΔR^2		.24	.29		.159	.209

Notes:
1. PP = Pair programming; CI = Continuous integration; REF = Refactoring; UT = Unit testing; DCOMP = Decision comprehensiveness; CFEED = Customer feedback; CPART = Customer participation. Standardized variables were used. Standardized beta coefficients are reported.
2. n = 56.
3. † p < .10; * p < .05; ** p < .01; *** p < .001.

leader-rated team effectiveness ($F_{16,\,55} = 4.213$, $p < .001$). The ΔR^2 for the interaction terms was .29. Hypothesis 1b, which predicted that customer feedback would strengthen the positive association between pair programming and team effectiveness, was not supported (β = -.087, p = ns). Hypothesis 2b stated that customer feedback would strengthen the positive relationship between continuous integration and team effectiveness. This hypothesis was supported (β = .244, $p < .05$). As figure 2.4a indicates, continuous integration had a stronger influence on team effectiveness when customer feedback was high. Hypothesis 2c, predicting a significant interaction between customer participation and continuous integration, was not supported (β = .024, p = ns). The interaction between customer participation and refactoring was positive and significant (β = .127, $p < .01$), supporting hypothesis 3b. The interaction plot in figure 2.4b shows that refactoring had a steeper slope when customer participation was high. Hypothesis 4b predicted that customer participation would strengthen the positive relationship between debate and team effectiveness. The coefficient for the interaction term was negative and significant (β = -.188, $p < .01$), contrary to expectations. Figure 2.4c indicates that debate was an important predictor of team effectiveness when customer participation was low. Finally, the interaction between customer participation and decision comprehensiveness was positive and significant (β = .218, $p < .01$), supporting hypothesis 5b. The interaction plot in figure 2.4d shows that decision comprehensiveness had a stronger influence on team effectiveness when customer participation was high.

The interaction model (model 3) predicting team member-rated team effectiveness explained 32.7% of the variance ($F_{16, 55}$ = 2.670, $p < .01$). Adding the interaction terms to the main effects model yielded a ΔR^2 of .209. The pattern of results was similar to that of the model predicting leader-rated team effectiveness. The coefficients for the customer feedback x continuous integration (β = .184, $p < .05$), customer participation x refactoring (β = .077, p < .05), and customer participation x decision comprehensiveness (β = .142, p < .01) interaction terms were positive and significant, supporting hypotheses 2b, 3b, and 5b respectively. The coefficients for the customer feedback x pair programming (β = -.059, p = ns) and customer participation x continuous integration (β = .011, p = ns) interaction terms were non-significant. Thus, hypotheses 1b and 2c were not supported. Finally, the interaction between customer participation and debate was significant and negative (β = -.126, p < .01), contrary to hypothesis 4b. The interaction plots (figures 2.4e to 2.4h) were consistent with those for team effectiveness.

Discussion

The goals of this research were to: (a) theorize and test the relationship between the use of agile programming practices and team effectiveness, (b) develop and validate scales to measure the use of agile programming practices in software project teams, and (c) examine the mechanisms through which boundary spanning influences team effectiveness. A new set of measures was developed to capture the use of individual agile programming practices in software project teams. The main effects models for agile programming practices were found to significantly predict team effectiveness, explaining 19.3% of the variance in leader-rated team effectiveness and 11.8% of the variance in team-rated effectiveness. Finally, boundary spanning activities were found to moderate the relationship between the agile programming practices and team effectiveness, explaining an additional 29% and 20.9% of the variance in leader-rated and team-rated effectiveness respectively. The pattern of results was the same for both ratings of effectiveness.

Figure 2.4a

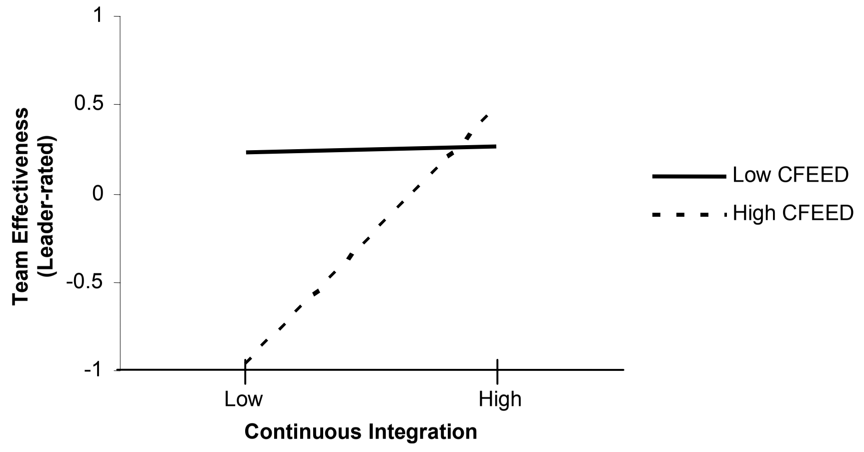


Figure 2.4b

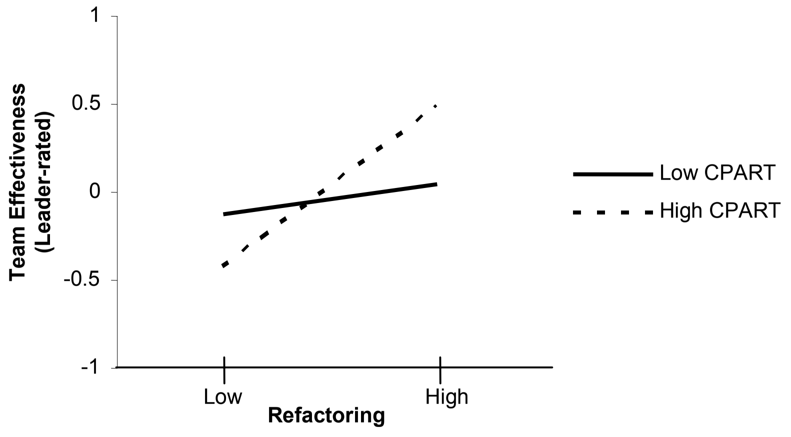


Figure 2.4c

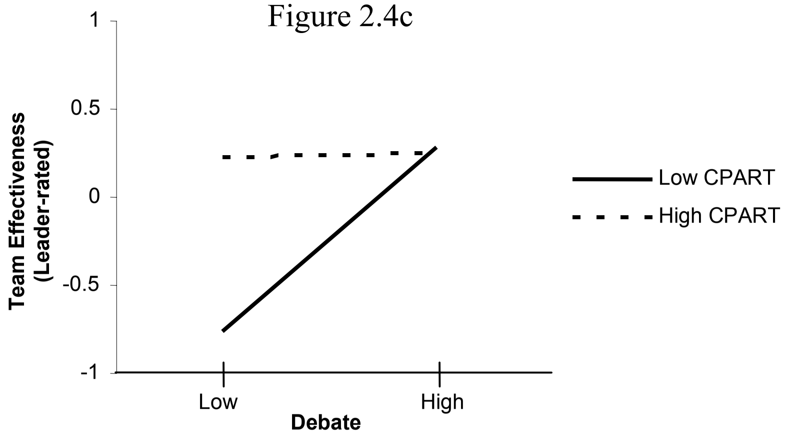


Figure 2.4d

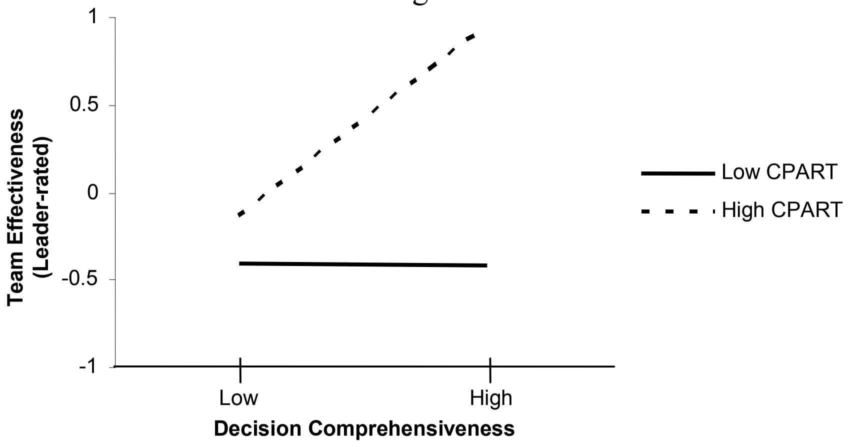


Figure 2.4e

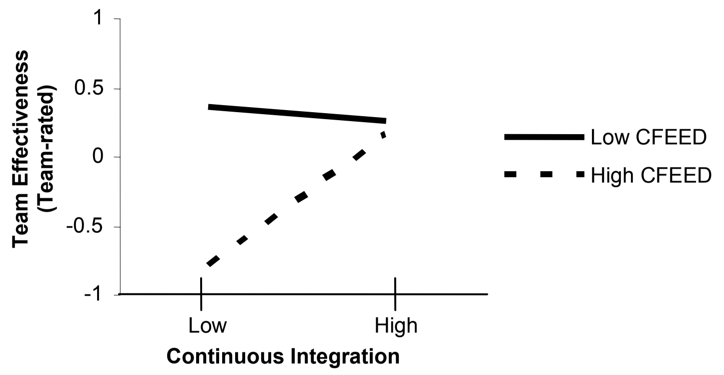


Figure 2.4f

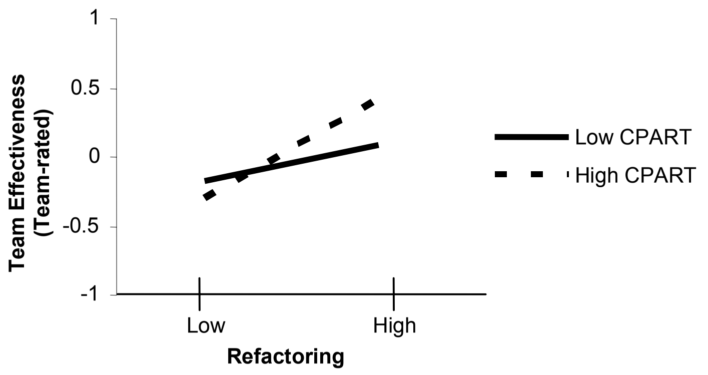


Figure 2.4g

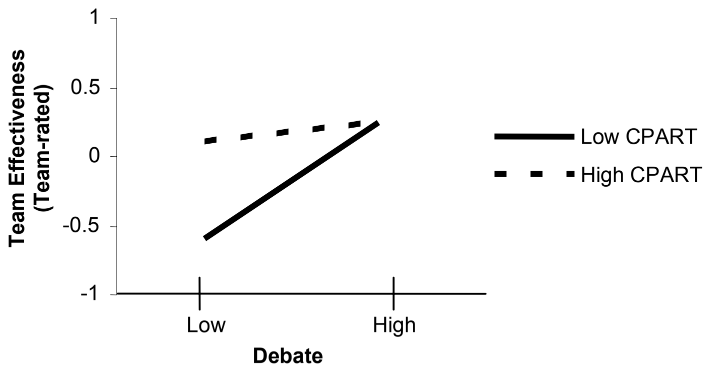
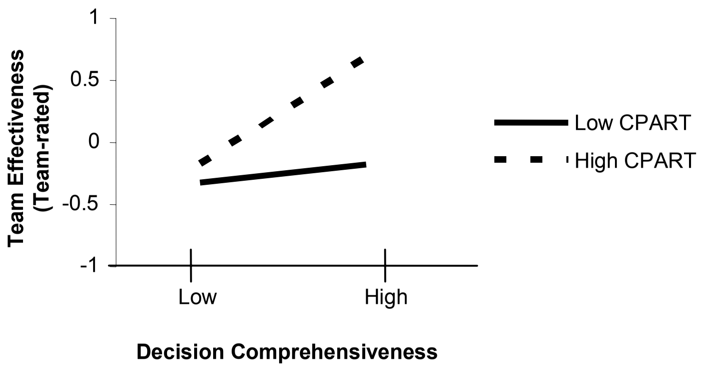


Figure 2.4h



Theoretical Contributions

This research makes several contributions to the literature. First, this research is one of the first to empirically examine the relationship between agile programming practices and software project team effectiveness. As noted earlier, the evidence to date has largely been anecdotal. Empirical evidence was found that links task-focused agile programming practices (pair programming, continuous integration, and refactoring) to software project team effectiveness. Empirical evidence was also found in support of a relationship between decision comprehensiveness in the planning process and software project team effectiveness. The results suggest that following these practices during software development has a positive influence on software project team effectiveness.

Second, as noted earlier, no scales currently exist to measure the extent to which software development teams use agile programming practices. In this research, a set of scales was developed, pilot tested, and empirically validated in a field sample of software development teams. The scales developed in this research have several benefits. First, the scales are focused at the level of the individual practices, rather than capturing a team's overall use of agile practices. This is important because software development teams differ in the extent to which they use various practices. Additionally, capturing the use of individual agile practices enables researchers to empirically evaluate which specific practices contribute to outcomes of interest. Finally, the scales are sensitive to levels of analysis issues. Specifically, the scales were developed using referent shift consensus, for which the software development team is the referent (Chan 1998). This ensures that respondents report on the activities of the team as a whole. The scales demonstrated adequate team-level properties including high within-team interrater agreement and greater between-team variance than within-team variance (Bliese 2000).

Third, this research probed deeper into the mechanisms through which customer feedback contributes to software project team effectiveness. Prior studies of software development project teams (e.g., MacCormack et al. 2001) have hypothesized a main effect of boundary spanning on team effectiveness. Consistent with prior work, customer feedback was found to positively influence team effectiveness. However, an even greater proportion of the variance in project outcomes was explained by the interaction between boundary spanning and agile programming practices. Specifically, boundary spanning was found to improve the effectiveness of two agile practices: continuous integration and refactoring. This finding suggests a more intricate role for boundary spanning in the context of software development and, thus, opens up the blackbox linking customer feedback to software project team effectiveness.

Finally, this research differentiated two levels of boundary spanning. Whereas customer feedback involves the gathering of information with minimal customer involvement in team tasks, customer participation involves a tighter coupling between the team and its customer. Here, the team receives customer feedback by actively involving the customer in team activities; both task-focused and decision making. As the results of this study indicate, each form of boundary spanning has different implications for team activities. Customer feedback was found to strengthen the relationship between continuous integration and software project team effectiveness. A tighter form of coupling, through customer participation, was found to be important for refactoring and decision comprehensiveness. These findings suggest that software development teams should adopt a more targeted strategy for getting customer feedback.

Practical Implications

The results of this research offer several implications for software project managers. The interaction effects are of particular importance. First, the results suggest that the adoption of an iterative approach to writing and integrating software code is an important team process. By continuously integrating newly developed code into the project, software development teams afford themselves greater flexibility to address changes in requirements if necessary. Second, it is important for software development teams to find ways to simplify their code on an ongoing basis while at the same time ensuring that the software addresses user requirements. Like continuous integration, refactoring provides software development teams with greater flexibility and enhances their ability to address changing requirements. Although refactoring may not seem to directly contribute to progress in meeting project completion goals, engaging in this practice can reduce the cost of making changes to code at later stages of the development cycle. Third, where possible, software coding should be conducted by pairs of programmers. As the results of this study suggest, pair programming is critical to software project success. Code written by pairs of programmers has been shown to contain fewer errors than code developed by individual programmers (Nosek 1998). This is the most controversial of the agile programming practices—largely because managers view it as an inefficient use of intellectual resources. However, the benefits of this practice outweigh the costs; especially when one considers the cumulative costs of coding errors. Fourth, during the planning process, it is important for software development teams to consider multiple alternatives to meeting customer requirements. By mapping out a comprehensive set of alternatives, software development teams are able to choose the best option. In addition, identifying alternative courses of action gives teams options that can be pursued if unforeseen contingencies render a chosen alternative undesirable. Teams are more likely to adapt to changes if they have a "plan B." Finally, to the extent possible, software development teams should try to involve customers in their ongoing activities. Such involvement allows software development teams to perform tasks more effectively. Additionally, involving the customer in the decision-making process enables the team to identify more viable solutions.

Strengths and Limitations

This research has several limitations that should be acknowledged. First, the number of teams in the study is relatively low. Obtaining a larger sample size was difficult given that this was a field study, which involved multiple points of data collection. The sample size limitation is offset by the use of two independent outcome measures—leader- and team-rated effectiveness—in an organization. Further, all teams in the sample launched their projects at the same time. Obtaining such a sample is rare in field research. The sample size is in line with that of most field research (e.g., Ancona and Caldwell 1992; Faraj and Sproull 2000). Second, all data were collected using a single methodology—survey questionnaires—thus, raising concerns about common method bias (Podsakoff and Organ 1986). However, this concern is allayed by the longitudinal design of the study, in which the independent and dependent variables were measured at different points in time, as suggested by Podsakoff, MacKenzie, Lee, and Podsakoff (2003). Additionally, data for one of the outcome variables, leader-rated team effectiveness, were collected from team leaders, while data for the independent variables were collected from team members. Data for team-rated effectiveness were collected from multiple team members. An additional strength in the design of the study is that data on the independent variables were collected from multiple members of each team, thus, providing a more accurate reporting of team activities than would be the case if data were collected from a key respondent.

Directions for Future Research

The results of this research have several important implications for future research. The finding that individual agile practices contribute positively to team effectiveness highlights the importance of focusing on software development team processes. Specifically, the results emphasize the critical impact of iterative practices, like continuous integration and refactoring, on software project outcomes. Continuous integration and refactoring are particularly well suited for managing uncertainty. Whereas continuous integration offers the team opportunities to incorporate requirements changes into the code, refactoring ensures that the code remains reasonably amenable to changes. Clearly, the use of such practices is a necessary part of the current software development environment (Cusumano and Yoffie 1999; MacCormack et al. 2001). Future research should therefore consider these practices in studies of software development team effectiveness. Pair programming proved to be an important determinant of effectiveness for the teams in the study. The study of pair programming should be expanded to the domain of software maintenance, where research has found that making changes to software increases the number of errors in software enhancement (Banker and Slaughter 2000). The use of pair programming during software enhancement could reduce the incidence of coding errors. Integrating and refactoring one software enhancement at a time could further reduce the number of errors.

In this research, team effectiveness was measured with respect to management of the overall project. Project performance encompasses a variety of dimensions including time to completion, managing costs, and meeting customer requirements. Unfortunately it was not possible to measure these various aspects of project performance. It is unlikely that teams are able to optimize their performance on all aspects of project performance. In reality, trade-offs must be made between one set of performance parameters over another. For example, software project teams might have to pursue timely project completion at the expense of project quality or meeting all customer requirements. It is possible that the practices examined in this research are more effective for some performance parameters than they are for others. Future research should examine the impact of these agile practices on more fine-grained dimensions of performance.

The planning process during the design phase of the software project turned out to be an important contributor to team success. Specifically, being highly inclusive and integrative in the consideration of alternative approaches to designing the software yielded positive outcomes for the software development teams in this study. This is consistent with Simons et al. (1999) who found that decision comprehensiveness was positively related to top management team performance. In the context of the current study, decision comprehensiveness was especially fruitful when the customer actively participated in the planning and decision-making process. However, the measure of decision comprehensiveness did not differentiate between planning for requirements analysis versus system design. More project phase-specific measures are needed to capture the dynamics of the planning process as it unfolds over the life of a project. It would be important for future research to identify additional decision-making processes that improve software development team effectiveness. Although this research examined decision comprehensiveness during the planning stage, it would be equally important for future research to identify other stages of the software development process during which such decision-making would be beneficial. Marks et al. (2001) note that teams switch between planning and action phases throughout the life cycle of a project. The cyclical nature of these phases was not captured in the current research.

Boundary spanning was found to play a critical role with respect to the effectiveness of continuous integration, refactoring, and decision comprehensiveness. As prior research has found, gaining user involvement in system development yields favorable project outcomes (Barki and Hartwick 1994). However, this research uncovers the specific processes for which such user involvement is important. The results suggest that it is important to go beyond the simple effects of customer feedback. More complex models are needed to explicate the mechanisms through which customer feedback improves software project outcomes. In order to gain greater clarity on this complex relationship, it is imperative that future research studies further explore the role of various boundary spanning activities within the context of specific team processes. Such studies will shed light on what effective teams do with the feedback they receive.

Finally, the current research focused exclusively on the subset of agile practices that are task-oriented in nature. While the results lend support for the use of agile practices, it would be useful to know how the other facets of agile methods—e.g., coding standards, collective ownership, use of metaphor—influence software development team performance. Future research that examines the processes that such practices enable (or encourage) would be particularly useful from a policy/standard operating procedure (SOP) perspective. It would also be important for future research to examine how tools designed to support software development can be used effectively in conjunction with agile practices.

Conclusion

This research developed a model of agile programming practices and software development team effectiveness. Boundary spanning was argued to play a central role in enhancing the effectiveness of software development teams. A robust set of scales was developed to measure software development teams' use of agile programming practices. The model was empirically tested and supported in a longitudinal field study. To the author's knowledge, this is one of the first empirical studies to examine the impact of agile programming practices on software development team effectiveness. Further, this research is among the first to argue and find support for an expanded (complex) role for boundary spanning in software development teams. The results of the research model provide a deeper explication of the mechanisms through which customer feedback can be used to improve software project outcomes.

CHAPTER III

ESSAY 2: STRUCTURE VS. AUTONOMY: AN EXAMINATION OF CONTROL AND EFFECTIVE LEADERSHIP IN SOFTWARE PROJECT TEAMS

Abstract

A major challenge that managers face in leading software project teams is that of enabling teams to adapt to environmental contingencies—specifically those brought about by changing customer requirements. Before embarking on any project, software team leaders must make critical decisions about where decision authority lies, how project work will be rewarded, and what role to play in guiding team work. Such decisions are crucial for ensuring successful software project completion. However, an inherent tension exists between providing the autonomy needed for teams to remain adaptable to potential environmental contingencies and the structure needed to guide software development. Decisions about team governance have important implications for affective team outcomes. Further, affective outcomes are important antecedents of team effectiveness. Drawing on control theory, this essay examines the relationship between formal and informal control modes, team leader behaviors, and team outcomes. The relationships are tested in the context of a longitudinal field study of 56 software development teams. The results support the relationship between control modes and team outcomes. Further, leader behaviors are found to play a moderating role in the relationship between specific control modes and team outcomes. The implications of the findings for research and practice are discussed.

Introduction

Research in the software and information systems development (ISD) 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, Cooprider and Faraj 1998; Kirsch 1997; Sillince and Mouakket 1997). A critical point highlighted by research on software development is that project managers are in a prime position to influence development teams' progress towards achieving desirable project outcomes (Guinan et al. 1998; Kirsch 1997). While research on the management of software development projects has done much to improve our understanding of this complex creative process, there is still much we do not know about the effectiveness of leadership behaviors under various contingencies. One specific contingency, which continues to challenge project managers is that of changing customer requirements. Requirements changes are a challenge because they are nearly impossible to predict and they require significant team effort to address (Cusumano and Yoffie 1999; Iansiti and MacCormack 1997). Recent estimates suggest that the inability to manage changing requirements has resulted in up to $17 billion in project cost overruns (Standish Group 2003).

The information systems (IS) field has a rich literature on software project governance. Drawing on control theory (Ouchi 1979), prior IS research (e.g., Henderson and Lee 1992; Kirsch 1997; Kirsch, Sambamurthy, Ko and Purvis 2002) has outlined various governance mechanisms that should be used to manage IS development projects. In addition to outlining these governance mechanisms, Kirsch (1997) and Kirsch et al. (2002) have examined conditions that would lead to the use of certain governance mechanisms over others. However, Kirsch et al. (2002) point out that the effectiveness of these governance mechanisms on important project-related outcomes (e.g., project completion on time and under budget, satisfaction of project participants and customers) has not been examined. In particular, the implications of team governance decisions on project team affective outcomes have received little attention in the IS literature.

Given this backdrop, the goal of this essay is to examine and resolve the inherent tension between the need for structure in the software development process and the autonomy needed to respond to potential environmental contingencies caused by changing requirements. I accomplish this goal by examining the relationship between the modes of control outlined by Kirsch (1997) and software project team affective outcomes. In addition to examining the control modes and their influence on project team outcomes, I highlight the role of the project leader in enhancing software project teams' ability to manage potential environmental contingencies. I argue that this is a critical element that has not received much attention in the software development literature. In sum, my specific objectives in this essay are to:

- Develop theoretical links between formal and informal modes of control, and software project team outcomes;
- Identify team leader behaviors that can enhance software project team outcomes; and
- Empirically test the proposed relationships.

As noted earlier, the environment in which software projects are managed has become increasingly uncertain (Cusumano and Yoffie 1999; Iansiti and MacCormack 1997). However, the information systems (IS) literature provides little theoretical guidance on what project leaders can do to manage the software development process under such demanding conditions. Rather, much of the literature on software development project leadership has focused on team leadership under relatively stable conditions (Henderson and Lee 1992; Walz, Elam and Curtis 1993). Prior ISD research indicates that the exercise of control by project managers (leaders) is a significant contributor to the performance of software project teams (Henderson and Lee 1992; Kirsch and Cummings 1996). Within this literature *control* is defined as the set of mechanisms designed to motivate individuals to work in a manner that ensures the attainment of desired goals (Kirsch 1996). Recent work on control mechanisms suggests that ISD project managers exercise different modes of control in an effort to ensure successful project completion (Kirsch 1997). However, as stated earlier, the effectiveness of these control modes on team affective outcomes has not been examined. Yet these outcomes are critical to the ongoing effectiveness of project teams (Marks, Mathieu and Zaccaro 2001). Hence, a key contribution of this essay will be its examination of the effectiveness of leader-selected control modes on software project team affective outcomes.

Agile methodologies encourage the delegation of authority to software project 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 goals will be attained, and delegating responsibility to various team members. These characteristics represent critical elements of self-managing teams (Hackman 1986; Manz and Sims 1987). Therefore, in examining effective leadership of software project teams, this essay will draw on self-managing work team leadership research. The extant literature on self-managing teams suggests that a unique approach to leadership is required for such work groups (e.g., Courtright, Fairhurst and Rogers 1989; Druskat and Wheeler 2003; Manz and Sims 1987). The leadership of such teams proves to be especially challenging because leaders lack legitimate control over the actions of team members (Hackman 1986). Hence, it is of theoretical interest to understand the dynamic between specific leadership behaviors and various control mechanisms, and their impact on software project team outcomes.

Druskat and Wheeler (2003) recently identified four categories of effective self-managing team-focused leader behavior: relating, scouting, persuading, and empowering. The examination of these leadership behaviors in conjunction with leader-selected control mechanisms will make two contributions to the literature. First, there may be tradeoffs associated with various combinations of leadership behaviors and control mechanisms. For instance, the use of behavior control may not be effective if the project leader is enacting certain empowering behaviors. Hence, an examination of the interaction between control mechanisms and leadership behaviors would contribute to the literature. Furthermore, software project teams require a delicate balance between structure and autonomy and the study of interactions between control and leadership behavior would shed light on how such balance might be achieved. Second, the effectiveness of the leadership behaviors identified by Druskat and Wheeler (2003) has not been examined in teams that operate under conditions experienced by these software project teams (i.e., conditions of potentially changing requirements).

Research Questions

While the study of leadership in software project 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 ISD project managers continue to wrestle with is how to effectively manage team work in the system development process (Barki and Hartwick 2001). ISD is 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 development project managers must make important decisions about the appropriate governance mechanisms for managing both technical and social processes. Control mechanisms (Ouchi 1980) have been identified as being one of the most important antecedents of project team performance (Henderson and Lee 1992; Kirsch and Cummings 1996). In the context of ISD, control is defined as "attempts to ensure that individuals working on organizational projects act according to an agreed-upon strategy to achieve desired objectives" (Kirsch 1996, p. 1). For example, Henderson and Lee (1992) found that managerial and team member controls were positively related to IS design performance. Specifically, they found that managerial behavior control and team member outcome control were positively correlated with team efficiency, effectiveness, and time to project completion.

Much of the research on control mechanisms in ISD has focused on predicting the conditions, under which formal and informal controls will be used to manage the development process (e.g., Kirsch 1996, 1997; Kirsch et al. 2002). In a study of 32 system development projects, Kirsch (1996) found that behavior control and outcome control were determined by behavior observability and outcome measurability respectively. Her results also suggested that the use of self-control was dependent on outcome measurability and the project sponsor's domain knowledge. Further, Kirsch et al. (2002) found that behavior observability and client understanding of the ISD process negatively influenced the use of clan control. While this stream of research has done much to improve our understanding of what motivates the selection of various portfolios of control, there remains a need to investigate the impact of control modes on team performance. There is currently a paucity of empirical research examining the relationship between various control modes and software project team performance. Further, no research to date has examined the effectiveness of control mechanisms on the performance of software project teams using agile methodologies (henceforth referred to as agile software project teams). Hence, I pose the following research question:

Research Question 1: What is the relationship between formal and informal control modes and agile software project team effectiveness?

As noted earlier, agile software development teams possess many of the characteristics of self-managing teams including responsibility for a whole task, members with task-relevant skills, and discretion over the assignment of members to different tasks (Beck 2000; Cummings 1978). Therefore, the literature on self-managing teams provides a useful lens through which to study agile software project teams. Self-managing teams are characterized by high levels of autonomy (Hackman 1986). It may seem paradoxical, then, that self-managing teams require leadership in order to perform effectively. Prior research on self-managing teams has found that external team leaders play a critical role in ensuring team success (Druskat and Wheeler 2003; Kirkman and Rosen 1999). For instance, Manz and Sims (1987) found that effective team leaders gave self-managing teams the skills and resources to lead themselves. In other words, effective leaders enabled self-managing teams to manage leadership functions on their own. In their typology of leadership styles for self-managing teams, Stewart and Manz (1995) posited that empowered leadership promotes self-regulation, which consequently leads to long-term team effectiveness. Finally, in a recent qualitative study of self-managing teams, Druskat and Wheeler (2003) identified several behaviors that effective external leaders engaged in to enable their teams to achieve success.

Although agile software project teams possess many of the qualities of self-managing teams, the conditions under which they operate make them particularly unique. Agile software project teams are often faced with high levels of uncertainty as customer requirements and specifications change very quickly (MacCormack et al. 2001). Moreover, customer requirements tend to change frequently, thus, requiring software code to be flexible (Aoyama 1998). In addition, agile methodologies involve short cycle times, which puts pressure on software project teams to create functional units of software in short iterations (Beck 2000). Research has yet to examine how effective leaders enable agile software project teams to manage such environmental conditions. Little is currently known about what behaviors leaders of these teams engage in, to enable their teams to achieve success.

Research Question 2: What specific leader behaviors enable agile software project teams to perform effectively?

Finally, as noted earlier, very little is known about the nature of the relationship between control modes and leader behaviors and what impact the combined effects of these two factors might have on agile software project team outcomes. The selection of portfolios of control modes is often motivated by a variety of factors (Kirsch 1997; Das and Teng 1998). Research suggests that the effectiveness of these control mechanisms could vary as a function of the behaviors enacted by team leaders (Stewart and Manz 1995). I therefore pose the following research question:

Research Question 3: How do leader behaviors affect the relationship between control mechanisms and software project team outcomes?

The research questions stated above are motivated by a need to understand the complexity of the relationship between control mechanisms and project leader behaviors. Given the paucity of empirical research on agile software project teams, this essay is aimed at theorizing about the effective leadership and management of such teams and the subsequent impact on key affective outcomes and effectiveness.

The remainder of the essay is organized as follows. First, I review control theory, focusing specifically on its use in the IS literature. The literature on leadership in self-managing teams is also reviewed. An outline of the core characteristics of agile software projects will be followed by the development of specific hypotheses drawing on the control and leadership theories reviewed. An outline of the methodology for model testing including data collection, analysis, and results follows. I conclude by discussing the results and contributions of the essay.

Theoretical Background

Control Theory

Control is defined as the process of monitoring, evaluating, and providing feedback (Ouchi 1979). In order to understand the concept of control and its importance in organizational research, it is necessary to turn to the nature of organizations. Noted scholars on the theory of the firm argue that organizations exist to facilitate efficient economic transactions (Coase 1937; Williamson 1975). A fundamental problem of managing transactions between individuals is that their goals very rarely overlap (Barnard 1968). Hence, individuals' self-interest makes coordination of transactions especially problematic. However, such coordination is necessary as cooperation forms the basis for efficient transactions (Mayo 1945). Consequently, interdependence is a natural outgrowth of cooperative action between transacting individuals.

Organizational theorists have proposed two main governance mechanisms that facilitate the efficiency of transactions. In markets, transactions take place between individuals and are governed by a price mechanism (Coase 1937). Under the assumption that markets are perfect, the price mechanism assures the parties to the transaction that the exchange is fair. In fact markets represent the most efficient mechanism of control because prices convey all of the information necessary for an equitable transaction (Coase 1937). However, the effectiveness of the market mechanism is contingent on the ability to accurately measure and assign value to individual contributions to a transaction. As Williamson (1975) points out, it is not always possible to evaluate the true value of individual contributions to a transaction. In such cases, the market mechanism fails to protect individuals from opportunistic behavior (Williamson 1975).

In contrast to the market control mechanism, the bureaucratic control mechanism entails close monitoring and direction of subordinate behavior (Ouchi 1979; Weber 1947). According to Ouchi (1980) "each party contributes labor to a corporate body which mediates the relationship by placing a value on each contribution and then compensating it fairly" (p. 130). Control in this context is based on a social agreement on the legitimate authority of the hierarchy to perform this mediating role. In bureaucracies, the information necessary to assure equitable compensation is conveyed by formal rules or policies. Rules may specify the tasks to be completed and/or standards of output (Ouchi 1980; Weber 1947). Hence, equitable compensation to individuals is based on the extent to which they complete specified tasks or meet assigned output standards. One weakness of the bureaucratic mechanism is that supervisors' ability to provide control is dependent on a set of standards against which behavior or output can be compared (Ouchi 1980). Thus, when the standards of performance become ambiguous they cease to be a reliable source of information for equitable compensation and result in the failure of the bureaucratic control. This can happen when tasks become increasingly unique or highly integrated (Ouchi 1980).

Ouchi (1979) proposes a third control mechanism that provides efficiency when behavior or output performance is ambiguous. Clan represents an informal control mechanism that is built on social relations among subordinates. A manager faced with the challenge of ensuring that organizational goals are met without being able to clearly evaluate employee behavior (due, for example, to high task integration) or assess a meaningful standard of performance, is left with one other mechanism. As I suggested earlier, the reason that there is a need for control is the inherent existence of goal incongruence among parties to a transaction. Hence, when behavior or outcome performance is highly ambiguous, managers have the option of reducing goal incongruity so that organizational goals are met without a need for monitoring (Ouchi 1979). Ouchi (1979) suggests that managers encourage socialization processes that instill deep employee commitment to organizational objectives. Once individual goals are aligned with organizational objectives the need for surveillance is reduced. In contrast to the market mechanism, clan control is most efficient when output performance is highly ambiguous and opportunism is low (Ouchi 1980). Whereas markets rely on price to convey information, and bureaucracies rely on formal rules, the clan mechanism conveys information in the form of rituals and stories that highlight the values and beliefs of the organization (Clark 1970; Ouchi 1979).

In sum, the only social requirement for market control mechanisms is that there be a norm of reciprocity. A norm of reciprocity provides a disincentive for individuals to cheat in a transaction, for fear of inciting punitive actions from other members of the social system (Gouldner 1961). Price is the primary mechanism through which information about the equitability of the transaction is conveyed. Bureaucratic control requires a norm of reciprocity and agreement on legitimate authority for the organization to act as a mediator in the transaction (Blau and Scott 1962). Relevant information is conveyed through rules outlining appropriate behavior or standards of output to be achieved in order for employees to receive equitable compensation. Finally, the clan control mechanism requires a norm of reciprocity, legitimate authority, and shared values and beliefs. Members need to have a common agreement on the values and beliefs they espouse (Ouchi 1980). Relevant information is conveyed by way of traditions (e.g., rituals, stories). To the extent that shared values reflect organizational objectives, goal incongruence is reduced.

The use of controls has received a lot of attention in the IS literature, particularly in the context of ISD. Consistent with the organizational literature, IS researchers conceptualize control modes as being either formal or informal. *Formal* modes of control are viewed as a strategy for evaluating performance (Eisenhardt 1985; Kirsch 1997).

Behavior and outcome control, which are defined later, are classified as forms of formal control. Unlike formal controls, *informal controls* emphasize the social aspects of goal attainment. Social dynamics and self-regulation are the key mechanisms through which informal control is implemented (Eisenhardt 1985; Ouchi 1979). Two main informal controls—clan control and self-control—are defined later (Kirsch 1996; Ouchi 1980).

The use of formal and informal controls in ISD projects has received empirical validation in the IS literature. In a series of case studies, Kirsch and Cummings (1996) investigated the determinants of self-control by IS professionals who were involved in ISD projects. Kirsch and Cummings (1996) hypothesized that organizational tenure and task complexity would positively influence perceptions of self-control. As IS professionals spend more time in the organization, they become more knowledgeable about relevant tasks and are, thus, able to work autonomously, monitoring their own performance. Management may have a poor understanding of non-routine tasks and may therefore be unable to evaluate the performance of such tasks. In such cases, Kirsch and Cummings argue that IS professionals will invoke greater self-control. In contrast to organizational tenure and task complexity, formalization and hierarchical coordination were posited to negatively influence perceptions of self-control (Kirsch and Cummings 1996). Formalization, through the use of rules and policies, enables managers to control the completion of tasks through monitoring. Finally, managers are able to monitor employee behavior through the use of coordination mechanisms (Govindarajan and Fisher 1990). Hence, IS professionals are less likely to perceive a high degree of self-control as management oversight increases. The results of the study indicated that organizational tenure positively influenced perceptions of self-control. Contrary to expectations, however, Kirsch and Cummings (1996) also found that task complexity negatively influenced self-control while formalization positively influenced self-control.

Kirsch (1996) studied the emergence of behavior, outcome, clan, and self-control in 32 systems development projects. The emergence of these control modes was dependent on the amount of behavior observability, outcome measurability, and the extent of the controller's (IS project leader) knowledge about the process. Kirsch (1996) found a significant interactive effect of behavior observability and controller's knowledge on behavior control. Behavior observability and outcome measurability predicted outcome control, while outcome measurability and controller's knowledge predicted self-control. No effects were found for clan control.

Kirsch (1997) investigated the implementation of control modes in ISD projects. In particular, she examined the factors driving the implementation of specific combinations (portfolios) of control modes. Kirsch (1997) found that the process of portfolio construction involved the selection of appropriate formal control mechanisms, the design of new mechanisms for implementing formal control, and supplementing formal control mechanisms with informal control mechanisms. The choice of specific control mechanisms within the construction process, was driven by task characteristics, role expectations, and project-related knowledge.

More recently, Kirsch et al. (2002) examined the determinants of client liaisons' selection of control modes in ISD projects. They argued that behavior observability and outcome measurability would predict the type of control mode used by the client liaison. While high levels of outcome measurability were expected to lead to the use of outcome control, low levels were expected to lead to the use of self-control. High levels of behavior observability were posited to promote the use of behavior control when the client had an understanding of the ISD process (Kirsch et al. 2002). On the other hand, when clients had little understanding of the ISD process, clan control was selected. Finally, low levels of behavior observability were posited to lead to the use of self-control. The results of the study revealed a positive association between outcome measurability and outcome control. No interaction was found in the prediction of behavior control. However, behavior observability and client understanding of the ISD process contributed significantly to the prediction of behavior control. In the prediction of clan control, a significant interaction effect was found between behavior observability and client understanding of the ISD process. The negative coefficient for the interaction term indicated that when behavior observability was high, low levels of client understanding of the ISD process promoted the use of clan control. Finally, the results suggested a relationship between behavior observability and outcome measurability and self-control (Kirsch et al. 2002).

Finally, in one of the few studies to examine the relationship between control modes and ISD project performance, Henderson and Lee (1992) investigated the influence of managerial control and team member control on project performance. The results of their investigation suggested that managerial control was most effective when it was focused on behavior whereas team member control was most effective when it was outcome oriented (Henderson and Lee 1992). While Henderson and Lee (1992) found significant relationships between managerial and team member control and team performance, there is still much we do not know about the effectiveness of various control modes in uncertain environments. Further, it is unclear how team leader behaviors influence the effectiveness of control modes in improving team performance. In order to understand the implications of leader behavior for modes of control and agile software project team performance, I briefly review the literature on leadership in self-managing teams.

Leadership in Self-managing Teams

In their widely-cited paper on self-managing work teams, Manz and Sims (1987) ask, "If self-managing teams are truly self-managing, then why should an external leader be required?" (p. 106). Indeed, the concept of leadership in self-managing teams may seem somewhat paradoxical at first. However, team researchers have consistently argued for the importance of external leadership in self-managing teams (Hackman 1986; Kirkman and Rosen 1999; Manz and Sims 1984). Prior research has shown that effective leadership can make or break the success of a self-managing team (e.g., Cohen, Chang and Ledford 1997; Cordery, Mueller and Smith 1991). Research on self-managing teams has, thus, focused less on whether external leadership is necessary for the performance of such teams. Instead, more attention has been given to the examination of what effective leaders do to enable self-managing teams to perform well (Stewart and Manz 1995).

Self-managing teams have been associated with increased performance outcomes such as productivity and quality, as well as job-related outcomes like increased satisfaction and low absenteeism and turnover (Trist, Susman and Brown 1977; Versteeg 1990). In contrast to traditional work teams, self-managing teams have the autonomy to make decisions about group tasks, maintain task schedules, and assign members to different tasks (Hackman 1986). There is some variation in the extent to which self-managing teams can actually determine and regulate their own behaviors (Manz 1992;

Stewart and Manz 1995). The effective leadership of such teams has proven to be fairly challenging. Much of the challenge associated with leading self-managing groups has to do with role ambiguity of the team leader. The team is supposed to be autonomous and have the ability to manage itself—in essence performing the functions that normally would be done by management. However, the role of the leader is to provide the guidance and direction necessary to ensure that the team achieves organizational objectives (Stewart and Manz 1995). Empirical evidence suggests that traditional leader behavior can be detrimental to the performance of self-managing teams (Letize and Donovan 1990; Manz, Keating and Donnellon 1990). Clearly, the effective management of such teams requires some unique leadership behaviors (Manz and Sims 1987; Wall, Kemp, Jackson and Clegg 1986).

Prior literature suggests that leaders engage in practices that can range from purely autocratic to purely democratic (Bass 1990). Bass (1990) contends that autocratic practices include such leadership styles as directive, coercive, authoritarian, and production centered. On the other hand, democratic practices encompass leadership styles such as consensual, employee centered, relations focused, and consultative. Hence, while autocratic leadership emphasizes the sole possession of authority and control, democratic leadership is associated with shared authority and control (Stewart and Manz 1995). Leaders can also be passive or active in their involvement with team activities (Stewart and Manz 1995). Active leaders are highly involved in team activities and tend to be visible to team members. On the other hand, passive leaders are somewhat removed from the daily activities of the team.

Stewart and Manz (1995) developed a typology of leadership for self-managing work teams. The typology situates leadership styles in a two-by-two matrix. Overpowering leadership reflects active autocratic leaders that hamper the ability of teams to manage themselves (Lawler 1986; Stewart and Manz 1995). Passive autocratic leaders exhibit powerless leadership. Such leaders employ a laissez-faire approach to managing team activities, but are autocratic when expectations are not met. In contrast to powerless leadership, power-building leadership, which is exhibited by active and democratic leaders, provides guidance and skill development to teams. While power-building leadership has its benefits, it also hinders the autonomy of the team to manage itself. Active democratic leaders typically maintain control over behavior and team direction (Mills 1983; Stewart and Manz 1995). Finally, passive democratic leaders demonstrate empowered leadership. Such leaders truly enable self-managing teams to manage themselves. Many of the leader's functions are carried out by the team, including self-regulation and design of work processes (Hackman 1986; Lawler 1986; Manz and Sims 1987). Stewart and Manz (1995) suggest that such leaders are able focus on enabling team success by engaging in boundary spanning activities that reduce environmental uncertainty within the organization. It is this final type of leadership that has consistently yielded team effectiveness in studies of self-managing teams.

In their study of self-managing work teams, Manz and Sims (1987) identified 21 leader behaviors. Of these leader behaviors, actions that facilitated the ability of such teams to self-monitor, self-evaluate, and self-reinforce, were deemed the most important for effectiveness. The underlying theme for these behaviors was that effective leaders encouraged self-managing teams to practice leadership functions. Effective leaders encouraged teams to reinforce high group performance. They also encouraged teams to monitor and evaluate their own performance (Manz and Sims 1987). Kirkman and Rosen (1999) suggest that the autonomy associated with self-management is necessary but not sufficient for self-managing teams to be effective. Team empowerment, which encompasses autonomy, potency, meaningfulness, and impact, has been found to lead to increased team performance (Kirkman and Rosen 1999).

In a recent qualitative study of self-managing teams, Druskat and Wheeler (2003) examined the behaviors of effective leaders (i.e., leaders of high performing self-managing teams). Through a series of in-depth interviews and surveys, Druskat and Wheeler (2003) identified four key behaviors of effective leaders that were internally (i.e., focused on the team) and externally (focused on the organization) focused. Building on Yukl's (1989) taxonomy of leader behavior, the behaviors identified were: relating, scouting, persuading, and empowering. *Relating* involves the development of relationships with and a political and social awareness of the team. *Scouting* includes the search for information within the project team so as to assess its needs. Leaders demonstrate *persuading* behaviors when they work to influence team members to align their priorities with those of the organization. Finally, *empowering* behaviors involve delegating decision authority (Druskat and Wheeler 2003). I build on this recent conceptualization of effective leader behaviors, which draws on prior taxonomies and typologies of empowering leadership.

In the next section, I develop specific hypotheses about the relationship between various control modes and agile software project team performance and effectiveness. I also outline the logic behind the role of leadership behaviors as a moderator of the relationship between control modes and team outcomes.

Hypothesis Development

Figure 3.1 illustrates the proposed conceptual model for this study. Formal and informal control modes are hypothesized to influence team performance. Further, leader persuading and empowering behaviors are posited to moderate the relationship between specific control modes and team outcomes. The procedure for data collection and model testing follow.

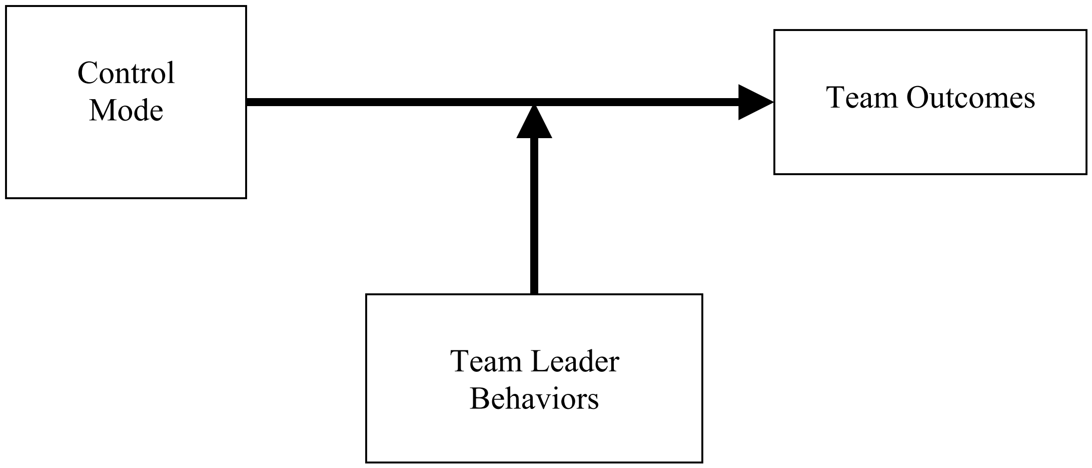


Figure 3.1: Conceptual Model of Control, Leadership Behavior, and Team Outcomes.

Formal Controls

Behavior control. Behavior control is synonymous with bureaucratic control and involves the articulation of specific rules and procedures to be followed (Kirsch 1997). Enacting specified behaviors is expected to yield desired project outcomes. Controllers (managers or team leaders) provide incentives by rewarding project teams on the basis of their adherence to the specified behaviors (methodologies). The exercise of behavior control requires that important software development processes be observable by the team leader (Kirsch 1997). In addition to this requirement, team leaders wishing to employ behavior control need to have an understanding of the software development processes being carried out. This is because incentives are structured such that teams are rewarded based on the extent to which they follow specified rules and procedures (Kirsch 1996; Ouchi 1979). Hence, it is important that the specific task-related details be clarified. Kirsch (1996) suggests that tasks in the software development process need to be programmable. Specific methods can be articulated for tasks such as software design, programming, testing, and documentation.

Agile software development processes have certain activities that are programmable. For instance, in the design phase of a software development project, teams need to use the planning game to outline a plan for how work will be accomplished and what the scope of the next release will be. Teams can draw out such plans by assessing business priorities and technical estimates (Beck 2000). Testing is one of the fundamental activities in agile programming. Both programmers and, where possible, customers write unit tests to ensure that one feature works flawlessly before moving on to another. Finally, rules can be articulated that specify the coding standards that programmers are required to adhere to (Beck 2000). These activities can be programmed into rules and procedures for the software development process. Behavior controls can be instituted by basing team rewards on the extent to which these observable activities are performed (Kirch 1996; Kirsch et al. 2002).

In order to efficiently manage a software project, project teams need information about customer requirements and necessary resources to make informed decisions about the appropriate behaviors to employ. As noted earlier, because customer requirements are often ill defined and are subject to change over time, agile software project teams are faced with high levels of uncertainty in the software development environment (Cusumano and Yoffie 1999; MacCormack et al. 2001). Thus, attempting to structure processes can yield detrimental effects for software project teams if they are unable to adapt their selection of activities. Pre-structuring software development activities through behavior controls may also have the unintended effect of focusing team members on performing specified behaviors, even when it is not appropriate. Prior research has shown that the standardization of processes under conditions of uncertainty often yields negative effects (e.g., Argote 1982). Nidumolu and Subramani (2004) have also recently echoed this view, arguing that the use of standardized methods decreases software development process performance. In addition, Stewart and Manz (1995) argue that the initiation of structures (behavior control) can disenchant members of self-managing teams as initiative and feelings of ownership over decision making are systematically damaged. Team members are likely to respond with compliance with specified procedures while harboring skepticism (Trist et al. 1977; Stewart and Manz 1995). Such reactions will likely result in lower team morale and a lack of commitment to team goals.

Hypothesis 1a: *Behavior control will negatively influence software project team effectiveness*.

Hypothesis 1b: *Behavior control will negatively influence software project team (i) morale and (ii) commitment*.

Outcome Control. In contrast to behavior control, outcome control does not focus on specifying appropriate rules and procedures for teams to follow. Instead, outcome control emphasizes the precise goals that are to be met by the software project team (Kirsch 1996). Software project teams are rewarded for meeting desired outcomes. Outcome control is less explicit about the processes used to achieve specific goals and is based on the ability to measure the outputs of task completion (Kirsch 1996; Ouchi 1979). Therefore, incentives and rewards are given based on the extent to which specified outcomes have been achieved. Managers of software project teams often set performance standards according to software quality (Kirsch 1997), meeting project deadlines (Henderson and Lee 1992), and managing project budgets (Nidumolu and Subramani 2004). While the performance of agile software project teams can be benchmarked against similar standards, there are some performance metrics that are unique to the agile programming environment. The literature on agile programming has consistently emphasized the importance of short releases (e.g., Beck 1999; Highsmith 1999; Cusumano and Yoffie 1999). The development of simple, yet functional, releases ensures that customers' most valuable business requirements are met in a timely manner (Beck 2000).

Outcome control provides benchmarks against which agile software project teams can monitor their performance. Outcome control enables teams to have greater latitude in determining the processes that will be used to achieve specified project goals (Kirsch 1996; Ouchi 1980). Such latitude allows agile software project teams to determine the appropriate activities to employ and when to employ them. For instance, project teams can decide how much unit testing is required and how often it is conducted. Further, the ability to structure programming activities as needed, enables teams to spend time on activities that will contribute to task completion.

In uncertain environments, the decentralization of decision-making authority has been shown to enhance teams' ability to respond effectively to unforeseen contingencies (e.g., Kirkman and Rosen 1997). Teams with such autonomy are better able to craft responses to environmental contingencies as needed (Edmondson 1999). In a recent study of 56 software development firms, Nidumolu and Subramani (2004) indicated a positive relationship between the use of standardized performance criteria (outcome control) and performance. Hence, outcome control is expected to positively influence project performance. Finally, the autonomy to determine work procedures and activities that the software project team will use to accomplish project tasks provides a sense of empowerment to team members (Kirkman and Rosen 1999). Team members with such autonomy are likely to develop a sense of ownership over ideas and, thus, become more committed to team decisions (Kirkman and Rosen 1997).

Hypothesis 2a: *Outcome control will positively influence software project team effectiveness.*

Hypothesis 2b: *Outcome control will positively influence software project team (i) morale and (ii) commitment*.

Although teams that are governed by outcome control may have a high degree of decision-making authority about how tasks are to be accomplished, they may not have the skills or competence to manage the leadership activities associated with such authority. Decision-making about task accomplishment often includes important activities such as mission analysis and strategy formulation and planning (Marks et al. 2001). Mission analysis involves the identification of the tasks to be accomplished, the evaluation of environmental conditions, and an assessment of the team's available resources. Strategy formulation entails the development of alternative courses of action that will lead to task accomplishment. Failure to perform these crucial activities has been shown to lead to poor performance (Gersick 1988).

Empowering behaviors enable team leaders to pass on their management skills to the team. According to Manz and Sims (1987), effective leaders are those that can lead teams to lead themselves. Team leaders accomplish this objective by continuously encouraging teams to self-monitor, self-evaluate, and self-goal-set. Stewart and Manz (1995) echo this view, suggesting that effective leaders first engage in power-building leadership by providing guidance and teaching skills. Once the team has learned self-management skills, active leadership is no longer required (Lawler 1986; Stewart and Manz 1995). Members of agile software project teams will be better able to manage the planning game—i.e., combine business priorities and technical estimates—by engaging in forward visioning. Further, the team can specify the use of metaphor to guide the software development process. Finally, empowerment may also afford the software project team the authority to manage task schedules, enabling developers to work reasonable hours. Empowering behaviors equip teams with the necessary tools to monitor and manage their progress toward task and goal accomplishment when outcome control mechanisms are in place. Thus, teams are better able to self-manage when leaders engage in empowering behaviors (Druskat and Wheeler 2003; Manz 1992; Stewart and Manz 1995). I, therefore posit a moderating effect of empowering behavior on the relationship between outcome control and software project team effectiveness, morale, and commitment.

Hypothesis 2c: *Empowering leader behaviors will moderate the relationship between outcome control and software project team effectiveness such that the relationship will become stronger with increasing empowering behavior.*.

Hypothesis 2d: *Empowering leader behaviors will moderate the relationship between outcome control and software project team (i) morale and (ii) commitment such that the relationship will become stronger with increasing empowering behavior.*

Informal Controls

Clan control. As mentioned earlier, clan control represents a form of group self-governance in which common values and beliefs are emphasized (Ouchi 1980). The emergence of clan control mechanisms is determined by the observability of behavior and the measurability of outcomes. Clan control is instituted when managers are unable to closely monitor behavior and the outcomes of team activity are difficult to objectively quantify (Ouchi 1980). Kirsch's research (e.g., Kirsch 1996; Kirsch et al. 2002) has emphasized that the extent to which controllers lack domain knowledge is also a key determinant of clan control use. Prior research on software development teams suggests that increased levels of self-control by teams can lead to improved performance (e.g., Bailyn 1984; Weinberg 1971). Autonomous teams are able to reallocate resources and use alternative work procedures to accomplish tasks without having to refer to management. Henderson and Lee (1992) posited a positive relationship between team self-control and performance. Although they did not find evidence of a relationship between team self-control and efficiency, they found a significant positive correlation between team self-control and effectiveness. Team autonomy and clan mechanisms encourage teams to reach consensus on key decisions during the software development process. This promotes a sense of commitment to the team and its mission, while also boosting morale.

Hypothesis 3a: *Clan control will positively influence software project team effectiveness*.

Hypothesis 3b: *Clan control will positively influence software project team (i) morale and (ii) commitment*.

Self control. When managers do not observe the day-to-day activities and behaviors of developers, leaders may encourage the exercise of self-control. Self-control is an individual governance mechanism. Individuals set their own task-related goals and then monitor, reward, and sanction themselves according to how well they meet their goals (Kirsch and Cummings 1996; Manz, Mossholder and Luthans 1987). Additionally, 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). Self-control builds on the intrinsic motivation of individuals. Thus, individuals are rewarded for how well they self-control (Manz and Sims 1987). When team members are given self control they are likely to gain more confidence in their abilities. However, within the team setting, such autonomy can have detrimental effects on the morale of the team. With everyone adopting their own approaches to managing interdependent tasks, there is likely to be much disagreement among team members regarding task accomplishment (Wageman 1995). Such disagreements often result in frustration among team members and, consequently, reduced team morale. Also, with team members pursuing their own interests, commitment to the team's goals will descrease. Further, left to their own devices, individual developers' goals may be highly incongruent, in which case development efforts might not converge to improve the software project team's effectiveness. While self control ensures that developers diligently self-regulate their behavior with regard to task accomplishment, there may be divergent approaches to the actual accomplishment of tasks. For instance, developers charged with programming functionalities into software may approach the coding process differently, resulting with incompatible modules. Such incongruence may result in reduced team project quality.

Hypothesis 4a: *Self control will negatively influence software project team effectiveness.*

Hypothesis 4b: *Self control will negatively influence software project team (i) morale and (ii) commitment*.

Druskat and Wheeler's (2003) framework suggests that team leader behaviors directly contribute to team effectiveness. As noted earlier, empowering behaviors enacted by team leaders provide team members with the necessary skills to effectively self-manage (Manz and Sims 1987). Persuading behaviors contribute to team effectiveness by ensuring that the team aligns its goals with those of the organization and providing greater mission clarity. Hence, persuading and empowering behaviors are expected to positively influence team effectiveness, team morale, and team commitment.

Method

The research model was tested within the context of a longitudinal field study. The study spanned four months and included three different points of measurement. In this section, I discuss the research setting, participants, measurement, and data collection process. A discussion of the analysis and results follows.

Research Setting and Participants. The participants were employees in a software development firm. Specifically, participants were all members of active software project teams. 73 teams agreed to participate in the study and a total of 509 employees in 56 software project teams provided usable responses at all three points of measurement for an effective response rate of 77%. Although it was desirable for all participating teams to provide responses at all measurement points, the duration of the study made this infeasible. 28% of the participants in the study were women. The average age of the participants was 29.4 with a standard deviation of 4.9. On average, the participants had 7.22 years of programming experience with a standard deviation of 2.54. The average team size in the study was 10 (range: 8 to 11)

Measures

The operational scales that were used in the research study are presented in Table 3.1. Interrater agreement (r_{wg}) was calculated, where appropriate, for individual-level responses to team-level variables in order to justify aggregation to a team-level score (James, Demaree and Wolf 1984). An average r_{wg} score of 0.70 and above is considered an adequate indication of within-team agreement on the rating of a scale. To ensure between-team variability on the constructs of interest, the intraclass correlation I (ICC[1]) was calculated. The ICC(1) determines how much of the variance in individual-level scale ratings can be explained by team membership (Bliese 2000). Bliese (2000) suggests that ICC(1) values between 0.05 and 0.25 are common.

Control mechanisms. Measures for the control mechanisms were adapted from Kirsch (1996). A 3-item measure was used to capture behavior control. Sample items included: “management expects the project team to follow an understandable, written sequence of steps toward the accomplishment of project goals," and "the project team is required to be familiar with existing written procedures and standard practices." The scale for outcome control included six items. Sample items for this scale included: "significant weight is placed upon project completion within budgeted constraints," and "project goals were outlined at the beginning of the project." Three items made up the clan control scale. A sample item is: "management attempts to understand the project team's goals, norms, and values." Finally, the scale for self control included three items. The Cronbach alphas for behavior control, outcome control, clan control, and self-control were .82, .83, .80, and .77 respectively.

Table 3.1: Measures Used in the Research Study.

CONSTRUCT	SOURCE
Control Mechanism:	
Outcome control	Adapted from Kirsch (1996)
Behavior control	Adapted from Kirsch (1996)
Clan control	Adapted from Kirsch (1996)
Self control	Adapted from Kirsch (1996)
Leader Behavior:	
Persuading	Adapted from Arnold et al. (2000)
Empowering	Adapted from Arnold et al. (2000) and Manz and Sims (1987)
Team Outcomes:	
Empowerment	Adapted from Kirkman et al. (2004)
Team morale	Adapted from Jehn and Chatman (2000)
Team commitment	Adapted from Jehn and Chatman (2000)
Team effectiveness	Adapted from Ancona and Caldwell (1992)

Leader behaviors. Several scales were used to measure leader behaviors. A six-item scale developed by Arnold, Arad, Rhoades, and Drasgow (2000) was adapted to measure leader persuading behaviors. The adapted scale included such items as: "the project leader explains company goals to our team," and "the project leader explains how our team's work fits into the company." The Cronbach alpha for this scale was .72.

Empowering behavior was measured using items adapted from Arnold et al. (2000) and Manz and Sims (1987). Sample items included: "the project leader encourages team members to solve problems together," "the project leader teaches team members how to solve problems on their own," and "the project leader encourages our team to monitor, be aware of, and to evaluate our level of performance." The empowering behavior construct had a reliability of .70. The average r_{wg} for persuading and empowering leader behaviors was .79 and .80 respectively. The ICC(1) for each construct was .29 and .20 respectively, indicating substantial variability across teams in the sample.

Team empowerment. Team empowerment was measured using a 12-item scale from Kirkman, Rosen, Tesluk, and Gibson (2004). The 12-item scale is an adaptation of Kirkman and Rosen's (1999) 26-item scale for team empowerment. It includes three items from each dimension of the empowerment scale (Kirkman et al. 2004). The team empowerment scale includes four dimensions: potency, meaningfulness, autonomy, and impact. Potency assesses the degree to which team members agree (or disagree) that their team has confidence in itself. A sample item is "My team believes that we can produce high quality work." Meaningfulness represents the extent to which team members believe that what their team does is valuable. Sample items include: "Our team cares about what it does" and "My team feels that its work is meaningful." Autonomy captures the extent to which the team has a lot of choice in what it does. A sample item is "Our team determines *how* things are done in the team." Finally, impact assesses the degree to which the team believes that what it does is important to the company (Kirkman and Rosen 1999). The impact scale includes such items as "Our team performs tasks that matter to this company" and "Our team makes a difference in this organization." The empowerment scale had a reliability of .79. The average r_{wg} for team empowerment was .77 and the ICC(1) was .27.

Team outcomes. Following Jehn and Chatman (2000), items from team satisfaction (3 items) and team cohesion (3 items) were adapted to measure team morale. Sample items are: "there is a group spirit within our team," and "overall I have enjoyed my experience with this team." The Cronbach alphas for the team satisfaction and team cohesion scale were .82 and .80 respectively. An eight-item scale was adapted to measure team commitment. The adapted scale included items such as "I am committed to this team," "I really care about the fate of this team," and "for me this is the best of all possible teams with which to work." The scale had a reliability of .75. The average r_{wg} for team morale (team satisfaction and team cohesion) and team commitment was .72, .79, and .83 respectively. The ICC(1) values were .12, .19, and .22 respectively. Finally, team effectiveness was measured using four items, including such items as "so far, we have done a good job on this project," and "the work that this team has done so far is of a very high quality." The reliability for this scale was .80. The average r_{wg} value was .77 and the ICC(1) was .19. A list of the adapted scales can be found in Appendix 3.A.

Procedure

Data were collected toward 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 large scale project involving 79 software project teams. The purpose of the project was to develop an integrated solution that would replace a client organization's existing legacy systems. The client organization's existing systems were fragmented across multiple business functions. The integrated solution was expected to provide greater cross-unit interoperability in the client organization. The project was expected to last four months.

Each of the 79 project teams was responsible for building a specific module that would be integrated into the final software solution. All project teams had a team leader who was responsible for managing progress toward project completion. The team leaders held weekly meetings with their respective teams to discuss project-related issues. Team leaders had a standardized set of performance criteria on which to rate their team. These standardized criteria were closely tied to how well client requirements were met and were frequently discussed at weekly meetings. Team leaders evaluated the output of the team as a whole and also evaluated individual developer performance. The organization valued the use of agile software development methodologies. These valued practices were outlined in white papers that were available to developers in the organization. However, team leaders had discretion over how much they enforced the use of these methodologies in software development projects.

All 79 project teams began working on the projects simultaneously. Participants were informed that the purpose of the study was to examine software project team processes. In order to organize team responses, all team members and team leaders were instructed to agree on a single team name, which would be used to track the questionnaires. Further, participants were instructed to provide their individual names and team name so that responses could be tracked over time. One week after the start of the projects, participants filled out the first questionnaire. Team members provided demographic information. Team leaders responded to questions regarding formal and informal control modes. Five weeks later, when the teams were well into their software

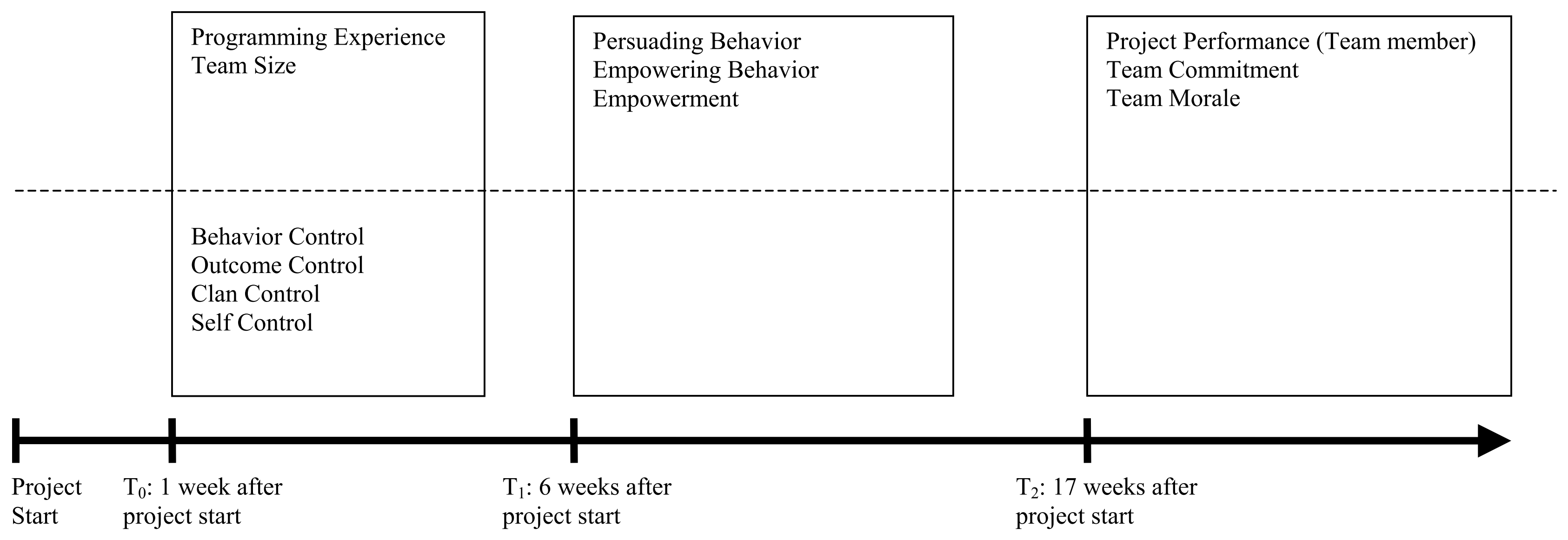


Figure 3.2: Summary of Study Design with Points of Measurement.

projects, the second questionnaire was administered. Team members responded to questions about their team leader's empowering and persuading behavior. 11 weeks later, the third and final questionnaire was administered. Team members responded to questions about team effectiveness, team commitment and team morale. Figure 3.2 presents the study design including all points of measurement.

Analysis

The convergent and discriminant validity of the outcome measures was evaluated using factor analysis (see Table 3.2). As evidence of convergent validity, all items loaded, as expected, on the pre-specified constructs. Further, all cross-loadings were less than .30 providing some evidence of discriminant validity. The construct correlations are displayed in Table 3.3. The research model was tested using moderated regression analysis. In the first step, control variables were entered into the equation. Next, the control mode, team empowerment, and leader behavior variables were entered into the model, followed by the interaction terms in the final step. Following Aiken and West's (1991) guidelines, the control mode and leader behavior variables were standardized and then used to create the interaction terms.

Results

Main Effects Hypotheses

The results of the regression analyses predicting team commitment, team effectiveness, and team morale are presented in Tables 3.4a and 3.4b. The main effects models (models 2) were used to test hypotheses 1a, 1b, 2a, 2b, 3a, 3b, 4a, and 4b. The results of the main effects models (models 2) predicting team effectiveness are reported

Table 3.2: Factor Analysis with Varimax Rotation.

	Factors									
Items	1	2	3	4	5	6	7	8	9	10
EFFECT1	**.71**									
EFFECT2	**.66**									
EFFECT3	**.79**									
EFFECT4	**.72**									
SAT1		**.88**								
SAT2		**.80**								
SAT3		**.85**								
COH1			**.88**							
COH2			**.78**							
COH3			**.73**							
COMM1				**.78**						
COMM2				**.70**						
COMM3				**.71**						
COMM4				**.66**						
COMM5				**.72**						
COMM6				**.71**						
COMM7				**.72**						
COMM8				**.74**						
BCONT1					**.70**					
BCONT2					**.76**					
BCONT3					**.74**					
OCONT1						**.72**				
OCONT2						**.75**				
OCONT3						**.79**				
OCONT4						**.72**				
OCONT5						**.66**				
OCONT6		.26				**.64**			.25	
CCONT1							**.70**			
CCONT2							**.72**			
CCONT3							**.74**			
SCONT1								**.75**		
SCONT2								**.79**		
SCONT3								**.82**		
PERS1									**.84**	
PERS2									**.93**	
PERS3									**.82**	
PERS4									**.91**	
PERS5									**.81**	
EMPR1										**.88**
EMPR2										**.82**
EMPR3										**.80**
EMPR4										**.80**

EMPR5	**.75**
EMPR6	**.73**
EMPR7	**.71**
EMPR8	**.75**

Notes:
1. EFFECT = Team effectiveness; SAT = Team satisfaction; COH = Team cohesion; COMM = Team commitment; BCONT = Behavior control; OCONT = Outcome control; CCONT = Clan control; SCONT = Self control; PERS = Persuading behavior; EMPR = Empowering behavior.
2. Loadings less than .25 not shown.

first, followed by results of the main effects models predicting team morale and team commitment.

The model predicting team effectiveness yielded a R^2 of .355 ($F_{9, 55} = 4.362$, $p < .001$). Hypothesis 1a predicted that behavior control would have a negative influence on team effectiveness. The beta coefficient for behavior control was non-significant ($\beta = .101$, p = ns). Thus, hypothesis 1a was not supported. Unlike behavior control, outcome control was hypothesized to positively influence team effectiveness. The beta coefficient for outcome control was positive and significant ($\beta = .458$, $p < .001$), supporting hypothesis 2a. For hypothesis 3a, which advocated a positive association between clan control and team effectiveness, the coefficient was positive and significant ($\beta = .187$, $p < .01$). Finally, hypothesis 4a predicted a negative relationship between self control and team effectiveness. As the results indicate, although the sign was in the expected direction, this hypothesis was not supported ($\beta = -.063$, p = ns).

Since the pattern of results for the regression models predicting the two facets of team morale—team satisfaction and team cohesion—was similar, only the results for team satisfaction are reported. As Table 3.4b indicates, the regression models explained 43.3% and 46% of the variance in team satisfaction ($F_{9, 55} = 5.676$, $p < .001$) and team

Table 3.3: Correlations, Means, and Standard Deviations.

Variables	Mean	SD	ICR	1	2	3	4	5	6	7	8	9	10	11	12	13
1. Effectiveness	5.21	1.47	.72	**.80**												
2. Satisfaction	4.99	1.07	.84	$.55^{***}$	**.82**											
3. Cohesion	4.55	1.27	.80	$.55^{***}$	$.99^{***}$	**.80**										
4. Commitment	4.19	1.33	.72	$.58^{***}$	$.60^{***}$	$.59^{***}$	**.75**									
5. Behavior control	4.44	1.21	.73	-.01	-.05	-.05	-.04	**.82**								
6. Outcome control	4.98	1.04	.72	$.30^{*}$	$.32^{*}$	$.32^{*}$	$.31^{*}$	.13	**.83**							
7. Clan control	4.16	1.09	.72	.22	$.24^{\dagger}$	$.24^{\dagger}$	$.23^{\dagger}$	-.04	-.12	**.80**						
8. Self control	4.43	1.19	.79	-.18	$-.24^{\dagger}$	$-.25^{\dagger}$	$-.23^{\dagger}$	.01	.00	-.09	**.77**					
9. Empowerment	4.00	1.17	-	.03	.00	.00	-.01	-.18	$-.23^{\dagger}$	.11	.08	**.79**				
10. Empowering	3.97	1.39	.78	.09	.12	.12	.11	-.19	$-.52^{***}$	.06	-.09	.07	**.72**			
11. Persuading	4.30	1.31	.86	.03	-.04	-.05	-.02	$-.33^{*}$	-.12	$-.24^{\dagger}$	$.24^{\dagger}$	.11	-.07	**.70**		
12. Team size	10	1.33	-	.02	.07	.07	.05	.07	-.15	-.04	-.06	-.06	.14	.05	-	
13. Avg. Prog. Exp.	7.22	2.54	-	.07	.00	.00	.02	.00	-.03	-.12	-.01	.19	.10	-.03	-.02	-

Notes:

1. ICR = Internal consistency reliability; Reliabilities (Cronbach-α) reported on the diagonal.
2. n = 56.
3. † $p < .10$; * $p < .05$; ** $p < .01$; *** $p < .001$.

Table 3.4a: Regression Analysis Predicting Team Commitment and Effectiveness.

	Commitment			Effectiveness		
	1	2	3	1	2	3
Controls:						
1. Team size	.064	.085	.118	.012	.019	.043
2. Avg. Programming Experience	-.002	-.032	.059	.041	.037	.103*
Independent Variables:						
3. Behavior control		.107	.215*		.101	.179**
4. Outcome control		.790***	-.242		.458***	-.289*
5. Clan control		.275**	.369***		.187**	.255***
6. Self control		-.124*	-.074		-.063	-.027
7. Empowerment		.133	.276***		.042	.145**
8. Empowering		.712***	-.021		.411***	-.120
9. Persuading		.241*	.367***		.202**	.294**
Interaction Terms:						
10. Empowering x Outcome control			.356***			.258***
Model F	.125	6.349***	12.666***	.142	4.362***	10.124***
df	53	46	45	53	46	45
R^2	.005	.554	.738	.005	.460	.692
Adjusted R^2	-.033	.467	.680	-.032	.355	.624
ΔR^2		.500	.213		.387	.269

Notes:
1. n = 56.
2. † p < .10; * p < .05; ** p < .01; *** p < .001.

cohesion ($F_{9, 55} = 6.215$, $p < .001$) respectively. Model 2 in Table 3.4a shows that the main effects model explained 46.7% of the variance in team commitment ($F_{9, 55} = 6.349$, $p < .001$). Hypothesis 1b argued for a negative relationship between behavior control and (i) team morale and (ii) team commitment. The coefficient for behavior control was non-significant in the model predicting team satisfaction ($\beta = .104$, p = ns) and the model predicting team commitment ($\beta = .107$, p = ns). Therefore, hypothesis 1b was not supported. For hypothesis 2b, which predicted a positive relationship between outcome control and (i) team morale and (ii) team commitment, the coefficient for outcome control was positive and significant (team satisfaction: $\beta = .634$, $p < .001$; team

Table 3.4b: Regression Analysis Predicting Team Morale (Team Satisfaction and Team Cohesion).

	Satisfaction			Cohesion		
	1	2	3	1	2	3
Controls:						
1. Team size	.040	.054	.083	.058	.078	.110
2. Avg. Programming Experience	.018	.001	.080	.003	-.024	.064
Independent Variables:						
3. Behavior control		.104	.198**		.106	.211*
4. Outcome control		.634***	-.264		.751***	-.247
5. Clan control		.234**	.316***		.265**	.356***
6. Self control		-.095†	-.052		-.117*	-.068
7. Empowerment		.090	.214**		.122	.260**
8. Empowering		.570***	-.068		.677***	-.032
9. Persuading		.223*	.332***		.236*	.358***
Interaction Terms:						
10. Empowering x Outcome control			.310***			.345***
Model F	.085	5.676***	12.293***	.112	6.215***	12.655***
df	53	46	45	53	46	45
R^2	.003	.526	.732	.004	.549	.738
Adjusted R^2	-.034	.433	.672	-.033	.460	.679
ΔR^2		.467	.239		.493	.219

Notes:
1. n = 56.
2. † p < .10; * p < .05; ** p < .01; *** p < .001.

commitment: $\beta = .790, p < .001$). The hypothesis was, therefore, supported. Hypothesis 3b argued for a positive relationship between clan control and (i) team morale and (ii) team commitment. The coefficient for clan control in the model predicting team satisfaction was positive and significant ($\beta = .234, p < .01$). The beta coefficient for clan control in the model predicting team commitment was also positive and significant ($\beta = .275, p < .01$). Therefore, hypothesis 3b was supported. Finally, hypothesis 4b suggested a negative association between self control and (i) team morale and (ii) team commitment. In the model predicting team satisfaction, the coefficient for self control was negative and marginally significant (β = -.095, p < .10). It is worth noting the coefficient for self control was negative and significant in the model predicting team cohesion (β = -.117, p < .05). Self control had a negative and significant effect on team commitment (β = -.124, p < .05). These results provide support for hypothesis 4b.

Moderation Hypotheses

The moderation hypotheses—hypotheses 2c and 2d—were tested by entering the interaction term (models 3) after the main effects models. Hypothesis 2c posited a moderating effect of empowering behavior on the relationship between outcome control and team effectiveness. Specifically, empowering behavior was expected to strengthen the outcome control-team effectiveness relationship. As model 3 of Table 3.4a indicates, the interaction term was positive and significant (β = .258, p < .001). Entering the interaction term into the regression equation resulted in a ΔR^2 of .269. A plot of the interaction (see Figure 3.3a) indicated a positive slope for the relationship between outcome control and team effectiveness when empowering behavior was high. In contrast, there was a negative slope for the relationship between outcome control and team performance when empowering behavior was low. The results supported hypothesis 2c. Hypothesis 2d predicted that empowering behavior would moderate the relationship between outcome control and (i) team satisfaction and (ii) team commitment. In the model predicting team satisfaction, the interaction term had a positive and significant beta coefficient (β = .310, p < .001; ΔR^2 = .239). Similarly, the interaction term had a positive and significant beta coefficient in the model predicting team commitment (β = .356, p < .001; ΔR^2 = .213). The interaction plots (Figures 3.3b to 3.3d) were consistent with the one for team effectiveness. Specifically, the relationship between outcome control and the outcome measures seemed to be positive when empowering behavior was high and negative when empowering behavior was low. Thus, hypothesis 2d was supported.

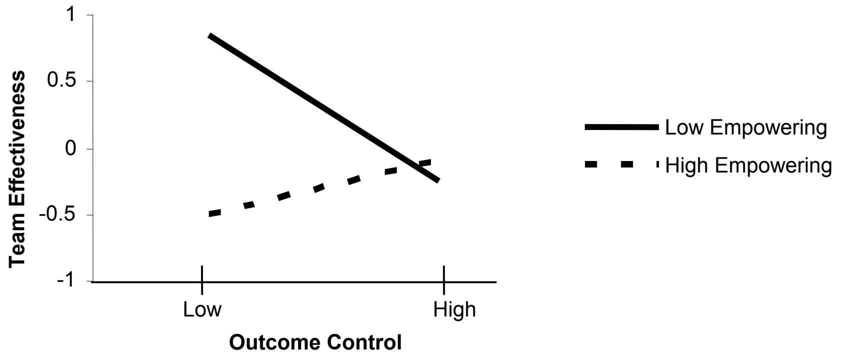


Figure 3.3a: Interaction Plot for Team Effectiveness.

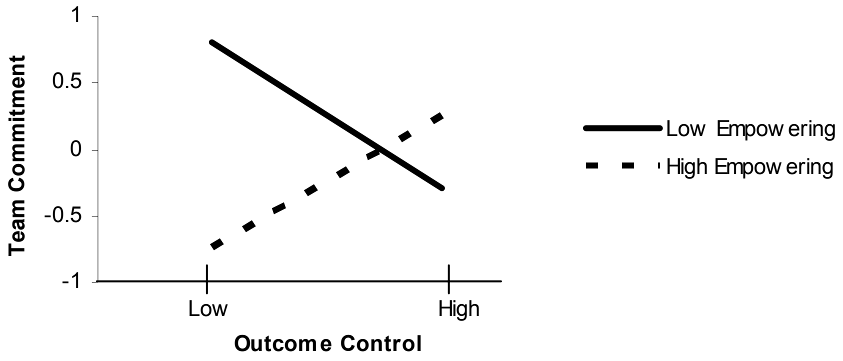


Figure 3.3b: Interaction Plot for Team Commitment.

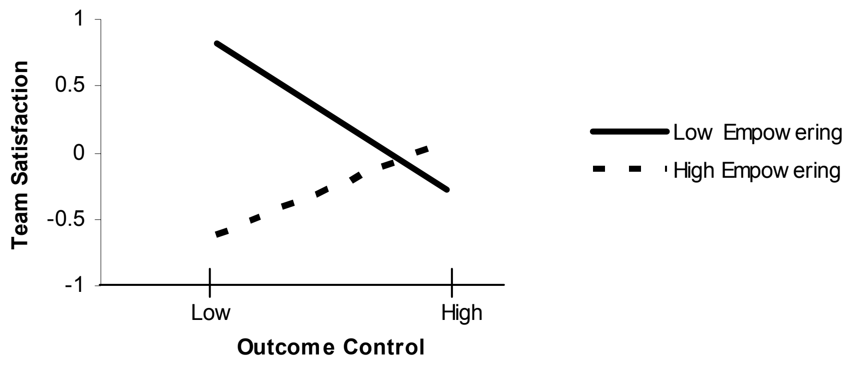


Figure 3.3c: Interaction Plot for Team Satisfaction.

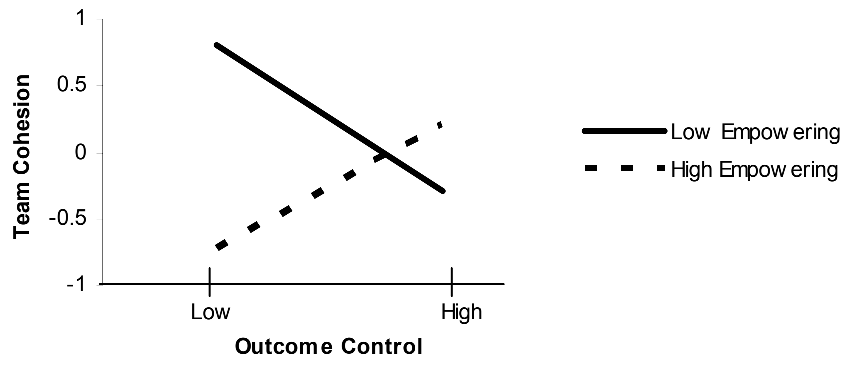


Figure 3.3d: Interaction Plot for Team Cohesion.

Discussion

The objective of this paper was to get a clear understanding of how software project team governance affects team outcomes. Specifically, this research sought to theorize the relationship between formal control mechanisms, informal control mechanisms, and leader behaviors; and team outcomes. While outcome control and clan control were posited to positively influence team outcomes, behavior control and self control were posited to have a negative influence on team outcomes. Further, empowering behaviors enacted by team leaders were posited to strengthen the positive relationship between outcome control and team outcomes. The results of this study largely supported these arguments. The main effects models explained 35.5%, 46.7%, 43.3%, and 46% of the variance in team effectiveness, team commitment, team satisfaction, and team cohesion respectively. Including the interaction term explained 62.4%, 68%, 67.2%, and 67.9% of the variance in team effectiveness, team commitment, team satisfaction, and team cohesion respectively.

Theoretical Contributions

This research makes several contributions to the software development team literature. First, this research theorized and examined the relationship between formal and informal control modes, and team outcomes. The results suggested that among formal control modes, outcome control yields favorable team outcomes with respect to effectiveness, commitment, and morale. Behavior control did not seem to have an influence on any of the team outcomes in the study. Among the informal control modes, clan control had a positive influence on team outcomes and self control had a negative influence. Collectively, these results suggest that control modes that are geared towards granting team autonomy over software project task accomplishment are especially effective. While outcome controls outline the performance criteria to be achieved and give software project teams the autonomy to decide how project tasks will be accomplished, clan control involves some degree of management intervention in which group norms and values are monitored. The results support prior research that has argued in favor of team autonomy as an important determinant of team effectiveness (e.g., Langfred and Moye 2004; Trist et al. 1977; Wall et al. 1986). Further, the findings of this research represent a response to a recent call by Kirsch et al. (2002) for research to examine the relationship between control modes and software project outcomes.

Second, support was found for the effect of two specific leader behaviors on team outcomes. Leaders' persuading behaviors, which influence the team by emphasizing the importance of aligning team and organizational goals, were posited to positively influence team outcomes. Leaders' empowering behaviors, which are geared towards enabling the team to manage itself, were also posited to positively influence team outcomes. The results indicated a positive relationship between persuading and empowering leader behaviors, and team outcomes. This study, therefore, provides an empirical validation of the leader behaviors identified by Druskat and Wheeler (2003) as important predictors of team outcomes. Druskat and Wheeler (2003) emphasized the importance of empirically validating the proposed relationships in a different context.

Finally, this research theorized and uncovered an important moderating relationship between outcome control and empowering behavior. Empowering behaviors enacted by team leaders were found to strengthen the relationship between outcome control and team outcomes. Thus, it seems that outcome control and empowering behaviors are complementary forces that enhance software project teams' ability to manage the delicate balance between the need for autonomy and the need for structure in software development work. The results highlight an expanded role for empowering behaviors outlined by Manz and Sims (1987) and point to an effective strategy in which software project teams are given the autonomy to decide how to approach team tasks and given the coaching necessary to manage task execution through important leadership functions such as monitoring progress toward team goals, self-regulation, and self-sanctioning.

Theoretical and Practical Implications

The results of this research have several implications for research and practice. The positive relationship between outcome control, clan control, and team effectiveness has major implications for research on software development. Both of these control mechanisms advocate some degree of team autonomy with respect to team process. These findings are consistent with Nidumolu and Subramani (2004) who argue in favor of the standardization of performance criteria and the decentralization of methods. Although Nidumolu and Subramani's (2004) findings are limited to performance, this research suggests that outcome and clan controls are important for team well-being (morale and commitment) as well. Such outcomes are arguably equally as important as performance, as they are central to effective team functioning (Marks et al. 2001). Given these findings, future research should consider the role of these control mechanisms in software development projects. Research that sheds light on the specific processes that these control mechanisms enable would be particularly valuable.

Self control was found to have a negative influence on team effectiveness. This presents an important dilemma for managers because software developers are typically compensated for their unique skills and abilities. 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, self control not only undermines software project teams' ability to perform effectively, it also diminishes team members' commitment to the team and its goals. Further, self control leads to decreased team morale. Managers are therefore encouraged to emphasize team outcome interdependence, rather than individual achievement, in software development project work.

The results of the current research also support an expanded role for software project team leaders. With a few notable exceptions (e.g., Guinan et al. 1998), the role of software development project leaders has been all but forgotten. The results of this research suggest that there are actions that software project team leaders can take to improve project success. Persuading behaviors help the team to understand its role in helping the organization to meet its goals. Such actions provide greater mission clarity for team members and foster an increased commitment to the team and its goals. Empowering behaviors are especially critical for effective team functioning. Team leaders are at their best when they teach their team members to lead themselves. Much of the literature on self-managing teams supports this view (Druskat and Wheeler 2003; Manz and Sims 1987; Stewart and Manz 1995). This research shows that team leaders can do a lot to ensure the success of their teams by enacting such behaviors. This is true of software development teams as well. This is especially true when teams have autonomy over task accomplishment. Future research that emphasizes an active role for software project team leaders is needed.

Future research should delve deeper into the role of team leadership and the behaviors that should be enacted over the life of the team. A recent theory of team coaching suggests that teams are amenable to different types of leader influence/interventions at different stages of their development (Hackman and Wageman 2005). Hackman and Wageman (2005) argue that coaching interventions that emphasize team effort, strategy, and skill are most likely to facilitate team effectiveness. Persuading and empowering behaviors represent two such interventions. This research did not emphasize the specific stage at which the enactment of persuading and empowering behaviors would be most effective. Arguably, persuading behaviors would be most effective during the early stages of the software project, the time during which the team is attempting to gain clarity on its mission and the task ahead. Empowering behaviors are more likely to be needed on an ongoing basis, although ideally teams should be self-managing as they near completion of the project. Future research is needed to understand the temporal dimension of effective team leadership.

Beyond emphasizing the importance of team leader behaviors, this research highlights the fact that software developers increasingly need to add collaboration and managerial capabilities to their repertoire of skills. Although software developers are typically hired for their technical expertise, successful team self-management requires that team members have the ability to manage all aspects of software project execution, including those functions that are typically performed by project managers. If software project teams are going to successfully exploit the autonomy they are granted, then software developers will need to be amenable to acquiring such skills. Future research should examine the relationship between the diversity of technical and managerial skill, and team effectiveness in software development teams.

Finally, this research did not explore the mediating mechanisms connecting control modes to team effectiveness. As noted earlier, it would be important for future research to examine the specific team processes enabled by different control modes. In order to gain greater clarity on the use of agile practices in software development teams, it would be important to understand which control mechanisms enable the effective adoption and use of such practices. Doing so would inform important organizational policy decisions governing the management and compensation of software project team work. As organizations experiment with how best to adopt agile practices, research that uncovers the policy-related implications will be increasing valuable.

Strengths and Limitations

This research has several strengths and limitations that must be acknowledged. First, this research 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. Future replications across multiple organizations are needed to validate the findings. Second, a longitudinal design was used, which involved three different points of measurement. Longitudinal field studies of teams are rare in the software development literature. Such a design alleviates concerns about common method bias, given that a survey methodology was used in this study (Podsakoff, MacKenzie, Lee and Podsakoff 2003; Podsakoff and Organ 1986). An additional strength of the study is that multiple respondents were used to gather data. While team leaders provided responses about formal and informal controls, team members responded to questions about team leader behaviors and team effectiveness. Multiple respondents were used to obtain this data, thus increasing the accuracy of the measures and further reducing concerns of common method bias.

Conclusion

This research sought to theorize and examine the relationship between control modes, team leader behaviors, and team effectiveness. These relationships were examined in the context of a longitudinal field study. Outcome and clan control were found to positively influence team effectiveness while self control was found to have a negative influence on team effectiveness. Persuading and empowering leader behaviors were found to have a positive relationship with team effectiveness and empowering behaviors were found to enhance the effectiveness of outcome control. The results of this research shed light on the role of control mechanisms and team leadership in supporting and promoting high team effectiveness.

CHAPTER IV

Conclusions

The main goal of this dissertation was to examine how software project teams can be better enabled to manage potential environmental contingencies—such as changes in project requirements. Flexibility was identified as a key factor that enables software project teams to remain adaptable to potential environmental contingencies during the software development process. It is no secret that in the dynamic competitive environment that currently exists, software project requirements are changing frequently. Such competitive conditions are likely to persist if not intensify in the years to come, thus, making this dissertation topical. This dissertation comprised two essays that present different perspectives on how flexibility can be enabled in software project teams. The first essay, which adopted a process-oriented view of the problem, pointed to specific software project team processes that would enable such flexibility. Specifically, I suggested that the use of agile programming practices, combined with boundary spanning activities, would contribute positively to software project team effectiveness. An empirical test yielded support for these contentions. The second essay in the dissertation focused on how the task environment, within which software project teams work, could be designed to provide the flexibility needed to respond to environmental contingencies during the software development process. Specifically, the second essay provided a team governance perspective by focusing on the control mechanisms used to guide software project team work. The hypotheses in the second essay posited a positive relationship between control mechanisms that promote team autonomy and software project team outcomes. Additionally, team leader behaviors that aligned team goals with organizational goals and enabled teams to self-manage were argued to play a central role in facilitating team outcomes. Empirical testing lent support for these hypotheses.

Collectively, the findings from the essays that make up this dissertation suggest that a two-pronged strategy can be used to enhance software project team effectiveness in software development. The first is focused internally on software project team processes. As the results in the first essay showed, the use of agile programming practices had a positive influence on team effectiveness. Further, continuous integration and refactoring—two agile practices that are specifically geared towards enabling process and code flexibility—were found to be even more effective when they were coupled with boundary spanning activities. The second part of the two-pronged strategy is focused on the task environment—specifically the policies used to guide team work. The results of the second essay emphasize the importance of policies that provide team autonomy in task accomplishment in software development projects. Such policies were found to be especially effective in facilitating software project team effectiveness when team leaders coached their teams to self-manage. Clearly, this two-pronged strategy is only a single step toward enabling teams to respond to potential environmental contingencies that can undermine effectiveness in software development projects. Only future replications in other organizations and the identification of additional contingencies will determine the relative efficacy of such a strategy.

The factors studied in the two essays may very well be complementary in nature. The limited sample size in this study made it difficult to examine an integrated model of team process and team governance factors that enhance team effectiveness. However, it is certainly plausible that these factors reinforce each other. Specifically, the structuring of the task environment—via control mechanisms—may be instrumental in enabling software project teams to effectively employ agile programming practices and boundary spanning activities. For example, clan controls might be used to influence teams' use of agile software development methodologies. Such an approach might be preferable to mandating the use of such practices through behavior controls. Additionally, some of the boundary spanning activities might be better facilitated if the team leader engages in empowering behaviors; as such behaviors would enable the team to benchmark its own progress vis-à-vis client expectations. Clearly, future work is needed to further examine how these governance mechanisms and internal team processes are related.

The results of this research provide some important general guidelines for enabling greater flexibility in software development. The agile practices that were examined represent specific practices that can be implemented to increase software project team flexibility. In addition, the control mechanisms and leadership behavior examined represent specific team governance mechanisms that can be put in place to facilitate greater software project team flexibility. The use of these practices should serve to improve software project team effectiveness.

End Notes

[1] URL: www.agilealliance.org

[2] r_{wg} is calculated as $\dfrac{J[1-\left(\dfrac{s_{xj}^2}{\sigma_{eu}^2}\right)]}{J[1-\left(\dfrac{s_{xj}^2}{\sigma_{eu}^2}\right)]+\left(\dfrac{s_{xj}^2}{\sigma_{eu}^2}\right)}$, where J is the number of items in the scale; s_{xj}^2 is the mean of the observed variances on the J items; and σ_{eu}^2 is the expected variance due to measurement error.

[3] ICC(1) is calculated as $\dfrac{MSB - MSW}{MSB + [(k-1) * MSW]}$, where MSB (mean square between) and MSW (mean square within) are obtained from a one-way ANOVA of individual level responses based on team membership; and k is the average team size.

[4] ICC(2) is calculated as $\dfrac{MSB - MSW}{MSB}$, where MSB (mean square between) and MSW (mean square within) are obtained from a one-way ANOVA of individual level responses based on team membership.

[5] The AVE is calculated as $\dfrac{\sum(\lambda_i^2)}{\sum(\lambda_i^2)+\sum(1-\lambda_i^2)}$

APPENDIX 2.A

Item Pools for Extreme Programming Practices

Pair Programming (Item Pool)

PP1. Most of the programming on this team is done by pairs of developers
PP2. Software developers on this team generally code software on their own, <u>without</u> a partner (r*)
PP3. On this team, we do our software development using pairs of programmers
PP4. We do pair programming a lot on this team
PP5. On this team, the coding of software is done by two developers sitting together at one computer
PP6. How often is pair programming used on this team? (Never; Very Frequently)
PP7. To what extent is programming carried out by pairs of developers on this team? (Never; Very Frequently)
PP8. How often do developers on this team code software on their own, <u>without</u> the presence of a partner? (Never; Very Frequently) (r)

Continuous Integration (Item Pool)

CI1. Each time a new unit is coded, members of this team integrate the software code
CI2. Members of this team integrate newly coded units of software with existing code
CI3. We combine new code with existing code on a continual basis
CI4. We <u>do not</u> spend time integrating new units of code with existing code (r)
CI5. Combining recently coded units of the project with other units of the code <u>is not</u> carried out by members of this team (r)
CI6. Our team <u>does not</u> take time to combine various units of code as they are developed (r)
CI7. How often do members of this team integrate new units of code with existing code? (Never; Very Frequently)
CI8. How often do members of this team combine newly coded units of software with the current software release? (Never; Very Frequently)

Refactoring (Item Pool)

REF1. Where necessary, members of this team try to simplify existing code without changing its functionality
REF2. We periodically identify and eliminate redundancies in the software code
REF3. We <u>are not</u> concerned with eliminating redundancies in the software code (r)
REF4. As long as the software performs as it should, members of this team <u>are not</u> concerned with simplifying the code (r)
REF5. We like to add flexibility to existing code so that new modules can be easily integrated later
REF6. Members of this team try to make the existing code flexible

REF7. On this team we try to optimize the efficiency of the code while ensuring that the software operates as it should
Unit Testing (Item Pool)

UT1. We run unit tests on newly coded modules until they run flawlessly
UT2. Members of this team actively engage in unit testing
UT3. On this team, coding additional functions does not proceed until existing code runs flawlessly
UT4. We code the next set of functions on a project <u>without testing</u> existing code (r)
UT5. To what extent are unit tests run by this team? (Never; Very frequently)

*r = reverse coded item.

APPENDIX 2.B

Measures adapted from existing scales

Leader-rated Team Effectiveness

1. I am satisfied with this team's work on this project so far
2. The work that this team has done so far is of a high quality
3. So far, we have done a good job on this project
4. The quality of the code we have developed so far is discouraging

Team-rated Team Effectiveness

1. We uncover many errors in our code during development and acceptance tests
2. We find very few errors during our development and acceptance tests
3. Development and acceptance tests suggest that the software project has contained very few errors up to this point
4. Development and acceptance tests indicate that our code contains numerous defects/errors

Customer Feedback

1. When design problems arise, we resolve them with the customer
2. We coordinate our activities with our client
3. Our team negotiates delivery deadlines and schedules with the customer
4. We review the project design with the customer

Customer Participation

1. We make an effort to involve the client in software development
2. We ask our client to participate in the coding process
3. Members of our team encourage the client to participate in testing code
4. We actively involve the client in our decision making activities
5. Whenever the team has an important decision to make, we ask the client to participate
6. On this team, we ask the client to help us to make decisions about the project design

Debate

1. In discussions about the project, team members state clear disagreements with each other
2. Different team members propose different approaches to the project
3. Team members openly challenge each other's opinions
4. Discussions about how to approach the project become heated

Decision Comprehensiveness

During planning...

1. To what extent does your team weigh multiple approaches against each other?
2. To what extent does your team examine the pros and cons of several possible courses of action?
3. To what extent does your team use multiple criteria for eliminating possible courses of action?

APPENDIX 3.A

Control Modes (Kirsch, 1996):

Behavior Control

1. Management expects the project team to follow an understandable, written sequence of steps toward the accomplishment of project goals
2. Management assesses the extent to which existing written procedures and practices are followed by the project team during the software development process
3. The project team is required to be familiar with existing written procedures and standard practices

Outcome Control

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

Clan Control

1. Management actively participates in project meetings to understand the project team's goals, values, and norms
2. Management places significant weight on understanding the project team's goals, values, and norms
3. Management attempts to understand the project team's goals, norms, and values

Self Control

1. Indicate the extent to which the tangible rewards given to the team by management, 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 the individual performance
3. Individual task performance is rewarded on this team

Leader Behaviors:
Persuading (Arnold et al., 2000)
Our project leader…:

1. explains company goals to the team
2. explains how our team's work fits into the company
3. explains the purpose of the company's policies to our team
4. explains rules and expectations to our team
5. helps team members focus on team goals

Empowering (Arnold et al., 2000; Manz & Sims, 1987)
Our project leader…:

1. suggests ways for our team to improve its performance
2. encourages team members to solve problems together
3. encourages team members to exchange information with one another
4. teaches team members how to solve problems on their own
5. encourages our team to be self-reinforcing of high group performance
6. encourages our team to be self-critical of low group performance
7. encourages our team to monitor, be aware of, and to evaluate our level of performance
8. encourages our team to set performance goals

Team Empowerment (Kirkman & Rosen, 1999):

Potency
My team…

1. Our team has confidence in itself
2. believes that we can produce high quality work
3. believes that we can be very productive

Meaning
My team…

1. Our team cares about what it does
2. Our team feels that its tasks are worthwhile
3. feels that its work is meaningful

Autonomy
My team…

1. Our team can select different ways to do the team's work
2. Our team determines as a team *how* things are done in the team
3. Our team makes its own choices without being told by management

Impact
My team…

1. Our team has a positive impact on this company's customers
2. Our team performs tasks that matter to this company
3. Our team makes a difference in this organization

Team Morale (Jehn & Chatman, 2000):

Cohesiveness

1. I generally like other members of my team
2. There is a group spirit within our team
3. I feel that I am really a part of this team

Satisfaction

1. Overall I have enjoyed my experience with this team
2. I am satisfied with my team
3. I have found my experience with this team to be enjoyable

Team Commitment (Jehn & Chatman, 2000):
1. I am committed to this team
2. I plan to continue working in this team
3. I talk up (brag about) this team to my friends as a great team to work on
4. I find that my values and the team's values are very similar
5. I am proud to tell others that I am part of this team
6. I really care about the fate of this team
7. For me this is the best of all possible teams with which to work
8. I am extremely glad that I chose this team to work with over other teams

Team-rated Team Effectiveness
1. I am satisfied with this team's work on this project so far
2. The work that this team has done so far is of a high quality
3. So far, we have done a good job on this project
4. The quality of the code we have developed so far is discouraging

References

Aiken, L. S., and West, S. G. *Multiple regression: Testing and interpreting interactions*. Sage, Thousand Oaks, CA, 1991.

Ancona, D. G. "Outward bound: Strategies for team survival in the organization," *Academy of Management Journal* (33:2), 1990, pp. 334-365.

Ancona, D. G., and Caldwell, D. "Bridging the boundary: External activity and performance in organizational teams," *Administrative Science Quarterly* (37), 1992, pp. 634-665.

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

Argote, L. "Input uncertainty and organizational coordination in hospital emergency units," *Administrative Science Quarterly* (27:3), 1982, pp. 420-434.

Arnold, J. A., Arad, S., Rhoades, J. A., and Drasgow, F. "The empowering leadership questionnaire: The construction and validation of a new scale for measuring leader behaviors," *Journal of Organizational Behavior* (21), 2000, pp. 249-269.

Bailyn, L. *Autonomy in the industrial R&D lab*. MIT Press, Cambridge, MA, 1984.

Banker, R. D., Datar, S., Kemerer, C. F., and Zweig, D. "Software complexity and software maintenance costs," *Communications of the ACM (36:11)*, 1993, pp. 81-94.

Banker, R. D., Davis, G. B., and Slaughter, S. A. "Software development practices, software complexity, and software maintenance performance: A field study," *Management Science* (44:4), 1998, pp. 433-449.

Banker, R. D., and Slaughter, S. A. "The moderating effect of structure on volatility and complexity in software enhancement," *Information Systems Research* (11:3), 2000, pp. 219-240.

Barki, H., and Hartwick, J. "Measuring user participation, user involvement, and user attitude," *MIS Quarterly* (18:1), 1994, pp. 59-82.

Barki, H., and Hartwick, J. "Interpersonal conflict and its management in information system development," *MIS Quarterly* (25:2), 2001, pp. 195-228.

Barnard, C. I. *The functions of the executive*. Harvard University Press, Cambridge, MA, 1968.

Basili, V. "Models and metrics for software management and engineering," *IEEE Catalog No. EHO-167-7* (p. 343): IEEE Computer Society Press, 1980.

Bass, B. M. *Bass & Stodgill's handbook of leadership: Theory, research, & managerial applications* (3rd ed.). Free Press, New York, 1990.

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

Beck, K. "Embracing change with extreme programming," *IEEE Computer* (32), 1999, pp. 70-77.

Beck, K. *Extreme programming explained.* Addison-Wesley, Reading, MA, 2000.

Bhattacharya, S., Krishnan, V., and Mahajan, V. "Managing new product definition in highly dynamic environments," *Management Science* (44:11), 1998, pp. S50-S64.

Blau, P. M., and Scott, W. R. *Formal organizations*. Scott, Foreman, San Francisco, CA, 1962.

Bliese, P. D. “Within-group agreement, non-independence, and reliability: Implications for data aggregation and analysis,” in *Multilevel theory, research, and methods in organizations,* K. J. Klein & S. W. J. Kozlowski (eds.), Jossey-Bass, San Fransisco, CA, 2000, pp. 349-381.

Chan, D. “Functional relations among constructs in the same content domain at different levels of analysis: A typology of composition models,” *Journal of Applied Psychology* (83), 1998, pp. 234-246.

Clark, B. R. *The distinctive college: Antioch, Reed, and Swarthmore*. Aldine, Chicago, IL, 1970.

Coase, R. H. “The nature of the firm,” *Economica* (4), 1937, pp. 386-405.

Cockburn, A. *Agile software development*. Addison-Wesley, Reading, MA, 2002.

Cohen, S. G., and Bailey, D. E. “What makes teams work?: Group effectiveness research from the Shop Floor to the Executive Suite,” *Journal of Management* (23), 1997, pp. 239-290.

Cohen, S. G., Chang, L., and Ledford, G. E., Jr. “A hierarchical construct of self-management leadership and its relationship to quality of work life and perceived work group effectiveness,” *Personnel Psychology* (50), 1997, pp. 275-308.

Constantine, L. L. *Constantine on people*. Yourdon Press, Englewood Cliffs, NJ, 1995.

Conte, S. D., Dunsmore, H. E., and Shen, V. Y. *Software engineering metrics and models*. Benjamin/Cummings Publication Company, Menlo Park, CA, 1986.

Cordery, J. L., Mueller, W. S., and Smith, L. M. “Attitudinal and behavioral effects of autonomous group working: A longitudinal field study,” *Academy of Management Journal* (34:2), 1991, pp. 464-476.

Courtright, J. A., Fairhurst, G. T., and Rogers, L. E. “Interaction patterns in organic and mechanistic systems,” *Academy of Management Journal* (32), 1989, pp. 773-802.

Cummings, T. “Self-regulated work groups: A socio-technical synthesis,” *Academy of Management Review* (3), 1978, pp. 625-634.

Curtis, B., Krasner, H., and Iscoe, N. “A field study of the software design process for large systems,” *Communications of the ACM* (31:11), 1988, pp. 1268-1287.

Curtis, B., Sheppard, B., Millman, M., Borst, A., and Love, T. “Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics,” *IEEE Transactions on Software Engineering* (5:2), 1979, pp. 96-104.

Cusumano, M. A., and Yoffie, D. B. “Software development on Internet time,” *IEEE Computer* (32:10), 1999, pp. 60-69.

Dansereau, F., Alutto, J. A., and Yammarino, F. J. *Theory testing in organizational behavior: The varient approach*. Prentice-Hall, Englewood Cliffs, NJ, 1984.

Das, T. K., and Teng, B. “Between trust and control: Developing confidence in partner cooperation in alliances,” *Academy of Management Review* (23:3), 1998, pp. 491-512.

Druskat, V. U., and Wheeler, J. V. “Managing from the boundary: The effective leadership of self-managing work teams,” *Academy of Management Journal* (46:4), 2003, pp. 435-457.

Edmondson, A. “Psychological safety and learning behavior in work teams,” *Administrative Science Quarterly* (44), 1999, pp. 350-383.

Eisenhardt, K. M. "Making fast strategic decisions in high-velocity environments," *Academy of Management Journal* (32:3), 1989, pp. 543-576.

Eisenhardt, K. M. "Control: Organizational and economic approaches," *Management Science* (31:2), 1985, pp. 134-149.

Faraj, S., and Sproull, L. "Coordinating expertise in software development teams," *Management Science* (46), 2000, pp. 1554-1568.

Fowler, M., and Highsmith, J. "Agile methodologists agree on something," *Software Development* (9), 2001, pp. 28-32.

Gladstein, D. L. "Groups in context: A model of task group effectiveness," *Administrative Science Quarterly* (29), 1984, pp. 499-517.

Gouldner, A. W. "The norm of reciprocity," *American Sociological Review* (25), 1961, pp. 161-179.

Govindarajan, V., and Fisher, J. "Strategy, control systems, and resource sharing: Effects on business-unit performance," *Academy of Management Journal* (33:2), 1990, pp. 259-285.

Guinan, P. J., Cooprider, J. G., and Faraj, S. "Enabling software development team performance during requirements definition: A behavioral versus technical approach," *Information Systems Research* (9:2), 1998, pp. 101-125.

Guzzo, R.A., and Shea, G.P. "Group performance and intergroup relations in organizations," in *Handbook for Industrial and Organizational Psychology,* M. Dunnett and L. Hough (eds.), Consulting Psychologists Press, Palo Alto, CA, 1992, pp. 269-313.

Hackman, J. R. “The psychology of self-management in organizations,” in *Psychology and work: Productivity, change, and employment,* M. S. Pallack and R. O. Perloff (eds.), American Psychological Association, Washington, DC, 1986, pp. 89-136.

Hackman, J. R. “The design of work teams,” in *Handbook of organizational behavior,* J. W. Lorsch (ed.), Prentice-Hall, Englewood Cliffs, NJ, 1987, pp. 315-342

Hackman, J. R., and Wageman, R. “A theory of team coaching,” *Academy of Management Review* (30:2), 2005, pp. 269-287.

Hargrove, R. *Mastering the art of creative collaboration*. McGraw-Hill, New York, NY, 1988.

Hartwick, J., and Barki,H. “Explaining the role of user participation in information system use,” *Management Science* (40:4), 1994, pp. 440-465.

Henderson, J. C., and Lee, S. “Managing I/S design teams: A control theories perspective,” *Management Science* (38:6), 1992, pp. 757-777.

Highsmith, J. *Adaptive software development*. Dorest House, New York, 1999.

Highsmith, J. “What is agile software development?” *Journal of Defense Software Engineering* (15:10), 2002, pp. 7-9.

Hunton, J. E., and Beeler, J. D. “Effects of user participation in systems development: A longitudinal field experiment,” *MIS Quarterly* (21:4), 1997, pp. 359-388.

Iansiti, M., and MacCormack, A. “Developing products on Internet time,” *Harvard Business Review* (75:5), 1997, pp. 108-117.

Ives, B., and Olson, M. H. “User involvement and MIS success: A review of research,” *Management Science* (30:5), 1984, pp. 586-603.

James, L. R., Demaree, R. G., and Wolf, G. "Estimating within group interrater reliability with and without response bias," *Journal of Applied Psychology* (69), 1984, pp. 219-229.

Jehn, K. A., and Chatman, J. A. "The influence of proportional and perceptual conflict composition on team performance," *International Journal of Conflict Management* (11:1), 2000, pp. 56-73.

Jones, C. *Software quality: Analysis and guidelines for success*. International Thompson Computer Press, London, UK, 1997.

Katz, D., and Kahn, R. L. *The social psychology of organizations*. Wiley, New York, 1978.

Kirkman, B. L., and Rosen, B. "A model of work team empowerment," in *Research in organizational change and development*, R.W. Woodman, and W.A. Pasmore (eds.), JAI Press, Greenwich, CT, 1997, pp. 131-167.

Kirkman, B. L., and Rosen, B. "Beyond self-management: Antecedents and consequences of team empowerment," *Academy of Management Journal* (42), 1999, pp. 58-74.

Kirkman, B. L., Rosen, B., Tesluk, P. E., and Gibson, C. B. "The impact of team empowerment on virtual team performance: The moderating role of face-to-face interaction," *Academy of Management Journal* (47:2), 2004, pp. 175-192.

Kirsch, L. J. "The management of complex tasks in organizations: Controlling the systems development process," *Organization Science* (7:1), 1996, pp. 1-21.

Kirsch, L. J. "Portfolios of control modes and IS project management," *Information Systems Research* (8:3), 1997, pp. 215-239.

Kirsch, L. J., and Cummings, L. L. "Contextual influences on self-control of IS professionals engaged in systems development," *Accounting, Management, and Information Technologies* (6:3), 1996, pp. 191-219.

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

Klein, K. J., Dansereau, F., and Hall, R. J. "Levels issues in theory development, data collection, and analysis," *Academy of Management Review* (19:2), 1994, pp. 195-229.

Klein, K. J., and Kozlowski, S. W. J. "From micro to meso: Critical steps in conceptualizing and conducting multilevel research," *Organizational Research Methods* (3:3), 2000, pp. 211-236.

Kozlowski, S. W. J., and Klein, K. J. "A multilevel approach to theory and research in organizations: Contextual, temporal, and emergent processes," in *Multilevel theory, research, and methods in organizations: Foundations, extension, and new directions,* K. J. Klein & S. W. J. Kozlowski (eds.), Jossey-Bass, San Francisco, 2000, pp. 3-90.

Kray, L. J., and Galinsky, A. D. "The debiasing effect of counterfactual mind-sets: Increasing the search for disconfirmatory information in group decisions," *Organizational Behavior and Human Decision Processes* (91), 2003, pp. 69-81.

Kraut, R., and Streeter, L. A. "Coordination in software development," *Communications of the ACM* (38:3), 1995, pp. 69-81.

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

Lawler, E. E. *High involvement management*. Jossey-Bass, San Francisco, 1986.

Letize, L., and Donovan, M. “The supervisor’s changing role in high involvement organizations,” *Journal for Quality and Participation*, (March), 1990, pp. 62-65.

Littlewood, B., and Strigini, L. “The risks of software,” *Scientific American* (267:5), 1992, p. 62.

MacCormack, A., Verganti, R., and Iansiti, M. “Developing products on ‘Internet Time’: The anatomy of a flexible development process,” *Management Science* (47:1), 2001, pp. 133-150.

Manz, C. C. “Self-leading work teams: Moving beyond self-management myths,” *Human Relations* (45), 1992, pp. 1119-1139.

Manz, C. C., Keating, D. E., and Donnellon, A. “Preparing for organizational change to employee self-management: The managerial transition,” *Organizational Dynamics (19:2)*, 1990, pp. 15-26.

Manz, C. C., Mossholder, K. W., and Luthans, F. “An integrated perspective of self-control in organizations,” *Administration & Society* (19:1), 1987, pp. 3-24.

Manz, C. C., and Sims, H. P., Jr. “Searching for the ‘unleader’: Organizational member views on leading self-managed groups,” *Human Relations* (37), 1984, pp. 409-424.

Manz, C. C., and Sims, H. P., Jr. "Leading workers to lead themselves: The external leadership of self-managing work teams," *Administrative Science Quarterly* (32), 1987, pp. 106-128.

Marks, M. A., Mathieu, J. E., and Zaccaro, S. J. "A temporally based framework and taxonomy of team processes," *Academy of Management Review* (26), 2001, pp. 356-376.

Maurer, F., and Martel, S. "Extreme programming: Rapid development for web-based applications," *IEEE Internet Computing* (6:1), 2002, pp. 86-90.

Mayo, E. *The social problems of an industrial civilization*. Division of Research, Graduate School of Business Administration, Harvard University, Boston, MA, 1945.

McCauley, R. "Agile development methods poised to upset status quo," *SIGCSE Bulletin* (33), 2001, pp. 14-15.

Mills, P. K. "Self-management: Its control and relationship with other organizational properties," *Academy of Management Review* (8), 1983, pp. 445-453.

Nidumolu, S. "The effect of coordination and uncertainty on software project performance: Residual performance risk as an intervening variable," *Information Systems Research* (6:3), 1995, pp. 191-219.

Nidumolu, S. R., and Subramani, M. "The matrix of control: Combining process and structure approaches to managing software development," *Journal of Management Information Systems* (20:3), 2004, pp. 159-196.

Nosek, J. "The case for collaborative programming," *Communications of the ACM* (41:3), 1998, pp. 105-108.

Nunnally, J. C., and Bernstein, I. H. *Psychometric theory*. McGraw-Hill, New York, 1994.

Ouchi, W. G. "A conceptual framework for the design of organizational control mechanisms," *Management Science* (25:9), 1979, pp. 833-848.

Ouchi, W. G. "Markets, bureaucracies, and clans," *Administrative Science Quarterly* (25:1), 1980, pp. 129-141.

Pfleeger, S. L. *Software engineering: The production of quality software*. MacMillan, New York, 1987.

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

Podsakoff, P. M., and Organ, D. W. "Self-reports in organizational research: Problems and prospects," *Journal of Management* (12), 1986, pp. 69-82.

Poole, C., and Huisman, J. W. "Using extreme programming in a maintenance environment," *IEEE Software* (November), 2001, pp. 42-50.

Reifer, D. J. "Requirements management: The search for Nirvana," *IEEE Software* (May), 2000, pp. 45-47.

Schulz-Hardt, S., Jochims, M., and Frey, D. "Productive conflict in group decision making: Genuine and contrived dissent as strategies to counteract biased information seeking," *Organizational Behavior and Human Decision Processes* (88), 2002, pp. 563-586.

Schweiger, D. M., Sandberg, W. R., and Rechner, P. L. "Experiential effects of dialectical inquiry, devil's advocacy, and consensus approaches to strategic decision making," *Academy of Management Journal* (32), 1989, pp. 745-772.

Sillince, J. A. A., and Mouakket, S. "Varieties of political process during systems development," *Information Systems Research* (8:4), 1997, pp. 368-397.

Simons, T., Pelled, L. H., and Smith, K. A. "Making use of differences: Diversity, debate, and decision comprehensiveness in top management teams," *Academy of Management Journal* (42), 1999, pp. 662-673.

Standish Group. The Standish Group report, 2003.

Stevens, J. *Applied Multivariate Statistics for the Social Sciences*. Lawrence Erlbaum Associates, Mahwah, NJ, 1996.

Stewart, G. L., and Manz, C. C. Leadership for self-managing work teams: A typology and integrative model," *Human Relations* (48:7), 1995, pp. 747-770.

Straub, D. W. "Validating instruments in MIS research," *MIS Quarterly* (13:2), 1989, pp. 147-169.

Swanson, K., McComb, D., Smith, J., and McCubbery, D. "The application software factory: Applying total quality techniques to systems development," *MIS Quarterly* (15:4), 1991, pp. 566-580.

Tesluk, P. E., and Mathieu, J. E. "Overcoming roadblocks to effectiveness: Incorporating management of performance barriers into models of work group effectiveness," *Journal of Applied Psychology* (84:2), 1999, pp. 200-217.

Tinsley, H. E. A., and Tinsley, D. J. "Uses of factor analysis in counseling psychology research," *Journal of Counseling Psychology* (34), 1987, pp. 414-424.

Trist, E. L., Susman, G. I., and Brown, G. R. An experiment in autonomous working in an American underground coal mine," *Human Relations* (30), 1977, pp. 201-236.

Tushman, M. L. "Special boundary roles in the innovation process," *Administrative Science Quarterly* (22), 1977, pp. 587-605.

Versteeg, A. "Self-directed work teams yield long-term benefits," *Journal of Business Strategy* (11:6), 1990, pp. 9-12.

Wageman, R. "Interdependence and group effectiveness," *Administrative Science Quarterly* (40), 1995, pp. 145-180.

Wall, T. D., Kemp, N. J., Jackson, P. R., and Clegg, C. W. "Outcomes of autonomous work groups: A long-term field experiment," *Academy of Management Journal* (29), 1986, pp. 280-304.

Walz, D. B., Elam, J. J., and Curtis, B. "Inside a software design team: Knowledge acquisition, sharing, and integration," *Communications of the ACM,* (36:10), 1993, pp. 63-77.

Weber, M. *The theory of social and economic organization*. (translated by A. M. Henderson & T. Parsons). Free Press, New York, 1947.

Weinberg, G. *The psychology of computer programming*. Van Nostrand Reinhold, New York, 1971.

Weinberg, G. M. *The psychology of computer programming*. Dorset House, New York, NY, 1998.

Williams, L. A., and Kessler, R. R. "All I really need to know about pair programming I learned in kindergarten," *Communications of the ACM (43)*, 2000, pp. 108-114.

Williams, L. A., Kessler, R. R., Cunningham, W., and Jeffries, R. “Strengthening the case for pair programming,” *IEEE Software* (14), 2000, pp. 19-25.

Williamson, O. A. *Markets and hierarchies: Analysis and antitrust implications*. Free Press, New York, 1975.

Yukl, G. A. “Managerial leadership: A review of theory and research,” *Journal of Management Development* (15), 1989, pp. 251-289.