THE IMPACTS OF AGILE DEVELOPMENT METHODOLOGY USE ON PROJECT SUCCESS: A CONTINGENCY VIEW

By

John F. Tripp

A DISSERTATION

Submitted to the
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Business Information Systems

2012

ABSTRACT

THE IMPACTS OF AGILE DEVELOPMENT METHODOLOGY USE ON PROJECT SUCCESS: A CONTINGENCY VIEW

By

John F. Tripp

Agile Information Systems Development Methodologies have emerged in the past decade as an alternative manner of managing the work and delivery of information systems development teams, with a large number of organizations reporting the adoption & use of agile methodologies. The practitioners of these methodologies make broad claims as to the benefits of their use. However, to date, only a few of these claims have been tested in the research literature.

Agile methodologies, including Extreme Programming, Scrum, and others, prescribe very different practices, some of which are contradictory. Additionally, the use of the practices of agile methodologies is not restricted to agile development projects, and has been observed in non-agile methodologies environments. Even so, the previous research literature has usually focused on practices prescribed by a particular agile method. So what is different about agile methodologies, and what is the appropriate lens through which to study them?

This dissertation finds that the most distinctive element of agile methodologies is their strong emphasis on obtaining and processing feedback from the environment. This dissertation evaluates the impacts of agile methodologies as indicated by the use of these feedback processes.

In this study, the theoretical lenses of team adaptation, organizational learning, and the prior literature on new product development are used to explain the importance

of a team's ability to process repeated and continuous feedback from the environment. We motivate hypotheses regarding the positive impact of agile methodology use on a multi-dimensional construct representing project success. This construct encompasses the quality of the delivered product, the benefits of the project to the organization, and impacts on project management outcomes. In addition, the nature of moderating influences of uncertainty on project success is discussed.

The research design for the study utilized a survey that collected responses from 83 agile development teams. Generalized linear modeling was used to test four hypotheses regarding the impact of the extent of agile methodology use on project success, and the moderating influences of uncertainty. It was found that agile methodology use positively impacts project success, while structural complexity negatively moderates the impact of agile use. It was also found that environmental dynamism positively moderated the impacts of agile methodologies on project success. Discussion of the results is provided.

To my mother, Pamela Richman Tripp, and to my father and step-mother John F. Tripp, Sr. and Asya Lapidus; you have always supported my endeavors, without ever criticizing. To my in-laws, John & Susan McLennan, for the support and encouragement that you show to your daughter's husband and his crazy endeavors.

To my children, Cayley Moriah, Joshua Lawrence Chamberlain, Zoe Elizabeth, Jonathan Edwards, Winston Churchill, Thomas Stonewall Jackson, and Normandy Reagan. Forgive me for being too busy to hear you, for missing soccer games, for missing too many dinners with you. Your love for your very imperfect father kept me going on many a long night's work.

Finally, to my wife, Molly Ann. No one is as incredible a help to her husband. No one loves a husband with more faults. No one is more courageous and self-sacrificing for her family. Without you, more than anyone else, my PhD could not have been completed. Thank you for marrying me, and thank you for the endless hours of putting the kids to bed alone, keeping the house alone, and teaching and instructing the children alone, all for the sake of my work.

I dedicate this dissertation to you, my family, who I love very much.

Soli Deo Gloria.

ACKNOWLEDGEMENTS

I would like to acknowledge the patience and guidance of my advisor, Dr. Vallabh Sambamurthy, who first implanted the idea of pursuing the PhD in my head, and who was forced to listen to me discuss the idea of this dissertation for far too long before it became a “good” idea. Without his guidance and inspiration, I would not have completed this dissertation.

Next, I would like to acknowledge the various faculty members who have, formally or informally, contributed to my completing the PhD process. First, I wish to thank the members of my committee, Dr. Roger Calantone, Dr. John Wagner, and Dr. Brian Pentland, I thank you for your guidance, feedback, correction, and encouragement. I could not have asked for a better set of faculty to assist me in my thinking and direction for this dissertation. I would also like to acknowledge the assistance of Dr. D. Harrison McKnight, who spent significant time helping me craft the survey instrument, and gave me help with this project whenever I asked, even though he was not a member of my committee.

I would also like to acknowledge the support and assistance given to me by Team Detroit, Inc., especially CFO Andy Weil and CIO James Weekley. Without the incredible flexibility afforded to me, and the support given to me during my first three years in the program, I could not have supported my family while also pursuing the PhD degree. I also thank many of my co-workers at Team Detroit who provided me with much of the inspiration for this project, and who contributed advice and feedback. I thank you all for your flexibility, enthusiasm, tolerance and support.

Finally, I would like to acknowledge the contribution and support of the agile development community, especially Ken Faw and John Birgbauer of Xede, Rich Sheridan of Menlo Innovations, Gary Gentry and Jay Aho of Pillar Technology, Eugene Keil and Chris Beale of Cengage Learning, Pat Reed of The Gap, and their many co-workers who helped me refine the ideas of this dissertation, and to identify avenues for data gathering. Also, I would like to thank the membership of the Mid-Michigan Agile Meetup, the Michigan Agile Enthusiasts, and the Chicago Agile Methodology Group for their encouragement and support.

Finally, I would like to acknowledge the many other people who lent support during this process, including the members of Providence PCA Church and Oakland Hills Community Church. Your encouragement meant a great deal.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter One: Introduction And Overview

"We are uncovering better ways of developing software by doing it and helping others do it." – The Agile Manifesto

Introduction

Information system development methodologies (henceforth, IS development methodologies) have been used over the past 50 years with the intent to both increase control and reduce the uncertainty and risk of IS development projects (Avison and Fitzgerald 1998; Hirschheim et al. 1995). IS projects have long been recognized as being difficult to define, manage, and deliver (e.g. - Barki et al. 2001; Keil 1995; Keil et al. 2000; Marakas and Elam 1998; Zmud et al. 1993). During its early years, software development was primarily an engineering concern. However, the systems development process context has increasingly become focused on the automation and support of core business processes. Therefore, development teams are being asked to adapt rapidly to changing technical, market, and user requirements (Conboy 2009), deliver business value more quickly and to respond to change more (Lyytinen and Rose 2006). However, even with the benefit of 50 years of method evolution, projects often exceed budget and time estimates, escalate for years, and regularly deliver far less than originally promised (Keil 1995).

IS development projects historically have been long-term efforts, delivering systems after months or even years. However, with the increasing pace of business competition, traditional development project timelines have been criticized as taking too long. Time pressures have escalated due to heightened competition and the escalating speed of technical innovation. This requires the software development process to deliver business value as quickly as possible. At the same time, software development

teams are being asked to adapt to rapidly changing technical, market, and user requirements. This pace of change must be managed without adversely impacting team performance.

Multiple software development methodologies have emerged in the last four decades of the twentieth century. At first, these methodologies emerged in response to the need to control and standardize the development process. Development methodologies continued to evolve to address issues related to growing systems and environmental complexity (Hirschheim et al. 1995). However, while these methodologies have evolved for more than 50 years, project failure rates continue to be extremely high (Standish Group, 2009). Project timetables continue to be long, flexibility is low, and the delivery of business value is usually achieved at the end of the project, if at all.

The latest generation of development methodologies, known as “agile” methodologies, have emerged over the past two decades (Beck and Andres 2004; Conboy 2009). These methodologies claim to be able to better handle the dynamic nature of the business environment (Boehm and Turner 2004; Cockburn 2001; Highsmith 2002; Schwaber 1996). Agile development methodologies share key philosophical underpinnings. These include the early and continuous delivery of software, the embracing and creating of change, the cultivation of empowered teams, and the delivery of simple solutions.

Agile development methodologies are different from traditional methodologies in several key ways. First, traditional methodologies usually planned to use 33% or more of the project duration in the planning phase (PMBOK 2000). In contrast, agile

methodologies recommend a maximum of 10% of anticipated project time to be spent on up front planning (Anderson 2004; Coad et al. 1999; Highsmith 2002), and most recommend much less. Second, most traditional methodologies usually planned to delivered the project system at the end of the project. In contrast, agile methodologies call for early delivery of business value through an iterative, evolutionary process. Because of this focus on early delivery, agile methodologies claim to increase the potential for business value to be delivered, while at the same time reducing the risk that changes in the environment will reduce the usefulness of the system before it is delivered.

Additionally, it has long been recognized that it is often difficult to properly define the requirements of the system up front (Brooks 1987). A great deal of research has been performed on various methodologies for better creating requirements specifications (Brooks 1987; Marakas and Elam 1998; Zmud et al. 1993). However, users often cannot clearly enumerate their requirements until they can compare working software with their expectations and task context (Brooks 1987). Traditional methodologies such as the waterfall methodology were designed to deliver software at the end of the project. Because of this, users often did not interact with the software until it was completed at the end of the project. Agile methodologies, because of their focus on early deliver of working software allow for incongruity between the developed software and users needs to be detected earlier in the project. Because of this, agile methodologies may be able to better adjust to meet the requirements that only emerge as users interact with the real system.

Because of these reasons, agile methodologies have been proposed by the practitioner community to deliver significant positive impacts to project performance. Claims regarding the impacts of the use of agile development methodologies fall into a few common categories. First, agile methodologies claim to boost productivity by creating a sustainable pace of development (Beck 1999), which creates an organizational environment in which people wish to work (Highsmith 2002). Second, agile methodologies claim to build trust between the software development team and their stakeholders due to the integration of stakeholders into the project team, and the regular and repeated demonstration of working software (Fowler and Highsmith 2001). Third, agile methodologies claim to better manage turbulent environments, due to their focus on interactive and adaptive design and delivery. Because of this iterative delivery paradigm, agile methodologies practitioners claim that agile methodologies deliver better and more useful software. This increased delivery success is claimed to reduce risk, and provide a better return on investment (Moran 2010) (Highsmith 2002).

These claims have spurred widespread agile methodology adoption, (Ambler 2009; Schwaber and Fichera 2005), with more than 50% of organizations reporting the adoption of agile methodologies (West and Grant 2010). Job postings referencing agile practitioners have increased more than 400% between 2008 and 2011 (Indeed.com 2011). Amongst practitioners, there is near unanimity as to the anticipated diverse set of benefits of agile development (see Table 1.1).

However, the normative claims of agile methodology impacts have been accompanied by little empirical evidence. The majority of reports of agile project success have been anecdotal (e.g., Abrahamsson et al. 2002; Lindvall et al. 2004;

Nerur et al. 2005), or focused on particular agile practice adoption (Cockburn and Williams 2001; Parrish et al. 2004; Williams et al. 2000). The few large sample empirical studies that have been reported focused on only some of the claims. For instance, the use of extreme programming practices has been found to be associated with higher code quality, as measured by a reduction in bugs (Maruping et al. 2009a).

Table 1.1. Anticipated Benefits of Agile Methodology Adoption (Version One 2010).

Anticipated Benefits	% Respondents Anticipating Benefit
Improved Project Visibility	88.7%
Enhanced ability to manage changing priorities	97.6%
Increased productivity	97.7%
Accelerated time to market	96.3%
Enhanced software quality	94.5%
Reduced risk	93.5%
Reduced cost	85.3%
Better management of distributed teams	54.2%
Simplified development process	89.6%
Improved alignment between IT and Business Objectives	88.6%
Improved Engineering Discipline	84.9%
Enhanced Software Maintainability	87.4%
Improved Team Morale	85.1%

Additionally, there are conflicting viewpoints as to how and when agile methodologies should be applied to the highly diverse contexts of software development projects. Some have claimed that agile development is only suited for small, non-critical projects (Agerfalk and Fitzgerald 2006; Boehm 2002). Others have reported successful use of agile development methodologies on projects that are of large scale, high technical complexity, and high coordination complexity (Anderson 2004; Beck and Andres 2004; Highsmith 2002; Schwaber and Beedle 2002). Since there is disagreement regarding the appropriateness and applicability of agile methodologies, it

is necessary that the research community develop a better understanding of how agile methodologies impact project performance.

Though there is a long tradition of research on the merits of alternative development methodologies, the research literature has only recently begun to seriously investigate the agile methodology phenomenon (Conboy 2009; Conboy and Fitzgerald 2004). Even as the research community has begun to devote significant resources toward the investigation of agile methodologies, this research is still at an early phase, and trails behind practice (Conboy 2009). Several issues have hampered the research literature. These include the inability to consistently define what “agility” means (Conboy 2009), and the limited conceptualization of agile development as being proxied by a particular agile methodology such as Extreme Programming, an individual engineering practice such as pair programming, or being viewed as an homogenous “black box” (Lee and Xia 2010).

Agile development methodologies cannot be treated as a “black box”. While agile methodologies share a common philosophical foundation, each stresses a different set of engineering and process practices. Research on agile development should not focus upon particular engineering practices for two reasons. First, these practices were used before the emergence of agile development methodologies, and can be used outside of agile methodologies. Second, the practices within agile methodologies are sometimes inconsistent and can even be contradictory. For example, collective code ownership is a practice that states that any member of the team can modify any portion of the code as necessary. However, while collective code ownership is a foundational practice in Extreme Programming (XP), it is specifically rejected in Feature Driven Development

(FDD). Collective code ownership has been used as one indicator of the level of agile methodology utilization (Maruping et al. 2009a), but as both FDD and XP are considered agile methodologies use of this indicator as a measure of agile methodology adoption may be inappropriate. Because these practices are contradictory, but both appear in agile methodologies, there is a need for a richer conceptualization of agile development methodology adoption and use: one that extends beyond the use of particular agile methodology practices.

Background

Agile development methodologies are a subcategory of iterative development methodologies. These methodologies focus on shorter timetables for delivery and the ability to rapidly respond to change. They stress the importance of good coding techniques, and good coders, but also implement a broad set of practices meant to manage work production, team norms and interaction, team boundary management, and the response to external influences. An overarching argument behind adopting agile development is that, because software development is an uncertain process, and therefore cannot be predicted, change cannot be avoided and in fact should be expected (Schwaber 1996). Therefore teams must be unencumbered by unnecessary process and planning, and organized to adapt well to change in order to succeed (Beck & Andres, 2004; Fowler & Highsmith, 2001).

Practitioners have proposed a set of methodologies that they consider to be agile methodologies (Table 1.2 gives a partial listing), yet these methodologies prescribe a wide variety of conflicting and sometimes contradictory practices to deliver agility. As noted above, Extreme Programming (XP) and Feature Driven Development (FDD) are

both considered examples of successful agile methodologies, yet they prescribe contradictory practices.

Table 1.2. Significant Agile Methodologies and Proponents.

Methodology	Proponent Agile Manifesto Author(s)
Extreme Programming	Kent Beck, Martin Fowler, Ward Cunningham, Robert Martin
Feature Driven Development	Jon Kern
Scrum	Mike Beedle, Ken Schwaber, Jeff Sutherland
Adaptive Software Development	James Highsmith
Crystal	Alistair Cockburn
DSDM	Arie van Bennekum

Adding to the confusion about agile development is the fact that most of the practices proposed in agile are not novel, and can be practiced in non-agile as well as agile settings (Beck 1999; Highsmith 2002; McConnell 1996). While the practices are important, practitioner literature also stresses the importance of the adoption of an agile "philosophy" within the team, management, and stakeholders as being at least as important to the development of an agile "ecosystem" (Highsmith 2002). Unfortunately, both practitioner and academic research has focused primarily on engineering practices when measuring agile methodology use. This has led to the criticism of agile methodologies despite their wide adoption, with many criticizing them for lack of rigor (Boehm and Turner 2004; Stephens and Rosenberg 2003), and others calling for their use to be limited to small, non-critical projects (Boehm 2002), or not used at all (Agerfalk and Fitzgerald 2006). However, multiple agile methodology practitioners report

that agile methodology practices are being used successfully on projects of all sizes and complexities (Anderson 2004; Beck and Andres 2004; Highsmith 2002).

For these reasons, there are important reasons to continue to research agile development. Since most agile practices are not novel, developing a richer conceptualization of the nature of agile software development methodologies is required. Since there is disagreement regarding the appropriateness and applicability of the use of agile methodologies, it is critical that the research community develop a better understanding of the how and the why of agile development methodologies, and when they are most impactful.

Purpose of this Study

As described above, the adoption of agile methodologies is reported to be widespread, but many of the normative claims have yet to be tested. Additionally, the methodologies that rest under the umbrella of “agile” methodologies are extremely diverse and sometimes contradictory in their prescribed practices. Further, the specific practices of most agile methodologies are not unique to agile environments. For these reasons, building a greater understanding of the salient components of agile methodologies is crucial.

Second, even if agile methodologies are adopted, there are still disagreements of whether or not the use of agile methodologies is appropriate in various contexts. It is reasonable to assume that influences from the environment, whether customer influences, management influences, corporate culture, or project constraints may influence the ability for agile methodologies to impact performance. Because of this,

developing an understanding of the key environmental and project characteristics that influence the success of agile projects is an important pursuit.

Therefore, the purpose of this research is to identify the broad components of agile methodology adoption and use, and to study when and how the use of agile methodologies impacts project success. To do this, a conceptual and operational definition of the use of agile methodologies is developed that captures multiple dimensions of the practice of agile methodologies, and integrate that new construct into a theoretical model of agile methodology impacts. Formally stated, the research question is:

How does agile methodology adoption and use impact project success?

Organization of Presentation

The remainder of the dissertation is organized as follows. Chapter Two presents a conceptual model that provides the theoretical underpinnings of the study with a review of the supporting literature in information technology, organization theory, and organizational behavior. The specific portion of this theory and the relationships that were investigated are discussed in Chapter three via the exposition of the research model, research questions and propositions. Chapter four describes the details of the research methodology that was used to investigate and answer the research questions. Chapter five provides an analysis of the results of the research study. Finally, Chapter six discusses the major findings of the research, implications for research and practice, and directions for future research, and concludes the dissertation.

Chapter Two: Theoretical Model and Supporting Literature

"All of us, as far as I can remember, thought waterfalling of a huge project was rather stupid." – Gerald W. Weinberg, Project Mercury Software Project Member

Introduction

The creation and use of development methodologies has been a core focus of the IS community for more than 50 years. During that time, a number of methodological philosophies and goals have emerged (Hirschheim et al. 1995), the latest of which is so-called "agile" development. A short recapitulation of the evolution of software methodologies will help to inform the discussion of agile development methodologies.

Hirschheim, Klein & Lyytinen (1995) described the evolution of seven generations of software development methodologies, and focus on the philosophy behind the methodologies and the environmental factors present at each stage. Development methodologies emerged out of a "Pre-Methodology Era" in the 1950s (p 29). As Hirschheim et al. (1995) note:

> *"At that time [the 1950s] the only conceivable system design task was programming and specifying computer room operations (Somogyi and Galliers 1987). To accomplish these complex tasks systems developers often followed a variety of systematic practices. New practices were invented as needed, and they were usually very technology oriented. Those practices which seemed to work…became the developer's 'rules-of-thumb' and, in a sense, his/her 'methodology'" (Hirschheim et al. 1995p. 29)*

In the "ad-hoc practices" of the 1950s and 1960s, development methodologies involved users speaking to programmers, after which programmers would build systems. From this pre-methodology era emerged more structured development methodologies in the 1970s. This transition was caused by the shift of computing

systems from being primarily designed for specialized and definable scientific applications, to being designed to support complex business and other real world processes (Avison and Fitzgerald 1998). However, even as methodologies evolved to manage the emerging issues of software development and engineering, software development projects continued to be marked by significant issues such as being considered incomplete, unstable, inflexible, and generally failing to meet the expectations of users and management (Avison and Fitzgerald 1998).

By the 1970's the state of software engineering was such that a "software crisis" was said to exist (Brooks 1987). Project estimates were grossly inaccurate, and projects were often delivered at as much as 100% over initial time and budget estimates or more. Methodologies and technologies were repeatedly proposed to be the magic bullet to "solving" the crisis. If teams would simply use a particular methodology, language, or programming paradigm, software projects would succeed. However, it was recognized that these methodologies, development standards, and other practices that had emerged did not assist in improving systems' essential functionality (Brooks 1987). In short, the "magic bullet" of software development methodologies was never found.

Table 2.1 lists seven generations of software development methodologies as described by Hirschheim et al. (1995). While the details of these generations are somewhat diverse, the repeated theme in the evolution is that methodologies emerged, were tried, and found to be lacking in some important way, which affected the underlying philosophy of development, and a new methodology was developed that was both responded to previous observation, and was in congruence with the emergent philosophy. Even so, after generations of software development methodologies, there is

a continued perception of a crisis in software engineering. U.S. Government audits, and private research agree that software projects are widely problematic, with late delivery, budget overruns and feature incompleteness being significant continued issues (Standish Group 2009).

The Evolution of Methodology - a Quest for Fit

As described above, the history development methodologies can be described as one of evolution and a quest for fit with environmental contingencies. In this dissertation, the impacts of agile methodologies on project success through the lenses of contingency theory, organizational learning, and team adaptation is described. The application of contingency theory has a long history within the IS field.

Contingency theory has been recognized as an important theoretical lens, and a building block for theoretical development in a variety of disciplines (Venkatraman 1989). Contingency theory in its simplest form states that the relationship between a cause (X) and effect (Y) is impacted by another variable, a contingency (C). Thus, the effect of X on Y will be different when C is low versus when C is high (Donaldson 2001). Contingency theory states that organizations recursively organize around creating a fit between internal structures, practices, and procedures, and the perceived contingent factors of the environment.

Table 2.1. Summary of Generations of Development Methodologies

Generation	Principal Management or Organizational Issue	View of Environment	Perception of Process
1. Formal Life-Cycle Approaches	Control of SDLC; guidance of analysts/ programmers through standardization	In equilibrium	Technical
2. Structured Approaches	Productivity (information requirements quality assurance to meet the '5Cs' – clear, concise, cost-effective, comprehensive and complete specifications); better maintainable systems; control of analysts/programmers (division of labor, e.g. Kraft, 1977)	In equilibrium	Technical, but with social consequences
3. Prototyping and Evolutionary Approaches	Speed and flexibility of development (SDLC methodologies take too long and are too rigid); overcoming analysts/use communication gap with technical specifications; emphasis is on getting the right kind of system vs. getting the system right	In equilibrium	Technical, but with social consequences
4. Socio-Technical, Participatory Approaches	Control of development by users through participation; conflict management in development; joint optimization: cost-effectiveness and better QWL through technology	In conflict, and in need of compromise	Joint technical and social process

Table 2.1. (cont'd)

5. Sense-Making and Problem Formulation Approaches	Dealing with multiple perspectives in problem framing; software development as social reality construction.	In conflict, and in need of compromise	Mostly a social process
6. Trade-Union Led Approaches	Labor/management conflict; workers' rights; industrial democracy.	In conflict due to class differences	Almost completely social, collaborative
7. Emancipatory Approaches	Overcoming barriers to effective communication due to power and social differentiation (e.g. blockage, bias, jargon, ambiguity); eliminating repression and furthering emancipatory effects of development (development as social learning and therapy, e.g. questioning dominant forms of thinking, improving access to facts and arguments, removing unwarranted uses of power, etc.)	In conflict, in need of rational compromise	Political

In the IS literature, the concept of fit has been applied at many levels, spanning from the organization strategy level (e.g. – IT strategy alignment) to the individual level (e,g, - task-technology fit). In this study, the view is utilized that contingencies impact at the business process level (Barki et al. 2001), and that particular environmental characteristics indicate that a particular set of process characteristics will deliver a better "fit", and potentially better outcomes.

Fit is an evolutionary process. Organizations attempt to fit their processes to the perceived environment, then modify their structure and practices when the fit between the structure and practice is less than satisficing (Donaldson 2001). This recursive structuring, feedback, and re-structuring is the hallmark of organizational learning (Argyris and Schoen 1978). Organizational learning requires the continual testing of the organization's "theory-in-use" by its members.

> *"They detect an error in organizational theory-in-use, and they correct it. This fundamental learning loop is one in which individuals act from organizational theory-in-use, which leads to match or mismatch of expectations with outcome, and thence to confirmation or disconfirmation of organizational theory-in-use" (Argyris and Schoen 1978). In this manner, as individuals detect a lack of fit between the environment and task, they make adaptation moves in order to regain fit with the new understanding of the environment."*

The concept of teams and adaptation will become a key lens for my analysis. One theory of team adaptation, team adaptability theory defines team adaptability as:

> *"a change in team performance, in response to a salient cue or cue stream, that leads to a functional outcome for the entire team. Team adaptation is manifested in the innovation of new or modification of existing structures, capacities, and/or behavioral or cognitive goal-directed actions." (Burke et al. 2006)*

In this definition three particular items regarding team adaptability should be noted: (1) adaptation is a response to feedback, (2) adaptation manifests in the

modification of team structure, routine and motivations, (3) adaption is goal-directed. Importantly, Adaptability is achieved by a recurrent cycle of planning, trial, learning, and feedback.

Finally, it should be noted that teams must adapt when the situations in which they are acting are uncertain, or unpredictable. Uncertainty has been characterized as the fundamental problem that faces any organization (Thompson 2003). The concept of uncertainty has been utilized widely in information and decision theory, in economic theory, and in organizational theory. While definitions of uncertainty in the information and decision theory approach the concept of uncertainty mathematically, this study adopts the definition of uncertainty simply as “the degree to which future states of the world cannot be anticipated and accurately predicted” (Pfeffer and Salancik 2003, p. 67). In the organization theory literature, uncertainty has been posited to arise from a number of sources. Lawrence & Lorsch (1967) described uncertainty as consisting of (1) the lack of clarity of information regarding the situation, (2) the length of time before receiving definitive feedback, and (3) the uncertainty of causality. Thompson (2003, p. 159) also viewed uncertainty as emerging from three sources, (1) the lack of ability to determine cause and effect in general, (2) the contingent nature of action, in which outcomes are partially dependent upon environmental elements and (3), interdependence of organizational components. Duncan (1972) described a similar three-component view of uncertainty as emerging from (1), the lack of clarity regarding how environmental factors influence the situation, (2) inability to predict risks of the outcomes of actions, and (3), the inability to confidently predict the likelihood of success of actions. Based upon these viewpoints, Duncan developed a model of uncertainty that

views uncertainty as a function of the complexity and dynamism within an environment. For this study Duncan's viewpoint is adopted, specifically that the complexity and dynamism of an environment are directly related to uncertainty - the inability to anticipate and accurately predict future states of the environment and organization.

The remainder of this chapter will develop these lenses, and explain how they provide a conceptual underpinning for the impacts of agile methodologies.

The remainder of the chapter is structured as follows. First, the phenomenon of agile development methodologies is described, including a discussion of software methodologies in general and a comparison of agile methodologies with traditional plan-based methodologies in particular is developed. Next, the theoretical lenses described above, are discussed, the role of team adaptation and learning is considered, and the specifics of agile philosophy and practices are presented within this discussion. After this, the concept of project success is addressed and the role of external contingencies is considered. Finally, a review of the literature on the impacts of agile development is presented.

What is a Software Development Methodology?

There has been long debate about whether “methodology” or “method” is the right term, what constitutes a methodology, and how practices and methodologies differ (Wynekoop and Russo 1995). Table 2.1 displays several selected definitions of method (or methodology)[1].

These definitions are consistent in describing that a development method emerges out of a philosophical view, defines certain practices to be carried out in the process of developing the system, and provides guidance for the organization and management of work. Taking the understanding that any method is more than the sum of its practices is key to understanding why agile methodologies are different from traditional methodologies.

Because of the importance of underlying philosophy in the motivation of any method’s practices, a comparison is made between the philosophy of agile development the philosophy of the earlier, traditional method, the structured, or waterfall method. Many have focused on engineering practices of specific agile development method. In contrast, the focus of this dissertation is on several key philosophical differentiators between the traditional and agile methodologies that are shared across all agile methodologies. The first of these is the concept of delivery – in other words, when does the project deliver a product, and how does that delivery enable teams to adapt to change.

---

[1] I will utilize the term “methodology” for the remainder of this dissertation.

Table 2.2. Selected Definitions of Software Development Method[ology]

Source	Definition
Avison et al.(1998)	"...a collection of procedures, techniques, tools, and documentation aids which will help the systems developers in their efforts to implement a new information system. A methodology will consist of phases, themselves consisting of sub-phases, which will guide the systems developers in their choice of the techniques that might be appropriate at each stage of the project and also help them plan, manage, control, and evaluate information systems projects...It is usually based upon some philosophical view, otherwise it is merely a method, like a recipe." (p. 8)
Hirschheim et al. (1995)	"An information systems development methodology is an organized collection of concepts, methods, beliefs, values, and normative principles supported by material resources." (p. 22)
Conboy (2009)	An development method encompasses the complete range of practices involved in the process of designing, building, implementing, and maintaining an information system, how these activities are accomplished and managed, the sequence and frequency of these activities, as well as the values and goals of all of the above. (p. 329)
Wynekoop & Russo(1995)	A systematic approach to conducting at least one complete phase (e.g. design or requirements analysis) of software production, consisting of a set of guidelines, activities techniques and tools, based on a particular philosophy of system development and the target system (p. 66)

Waterfall – Big Delivery

The Software Engineering Institute's Capability Maturity Model framework is currently the most accepted standard for measuring software engineering process excellence. It adopts the philosophy of the waterfall method, namely that the software development process is a definable process (Paulk et al. 1995; Schwaber 1996), and that software project performance can be predicted through the establishment of repeatable processes (Paulk et al. 1995). The classic waterfall model is represented in Figure 2.2. The waterfall method, characterized by the gate system familiar in physical engineering, had a variety of stage numbers and names, but all possessed aspects of the following:

- Analysis
- Design
- Construction
- Testing
- Deployment

The *analysis* stage consists of the identification of the business problem, the nature of the process being addressed by the project. The goal of the analysis phase is to identify the "facts" (Avison and Fitzgerald 1998) as they are understood, the details of the kind of data that had to be included in the system, requirements, constraints, exceptions, and the feasibility of the system. The outputs of the analysis phase of the lifecycle are detailed documents that describe the problem space, the requirements of the system, and the plan for completing the project. It is commonly understood that the

stages of the waterfall method do not overlap, and that backtracking is generally limited to moving one step back up the process (Boehm 1988).

The *design* stage(s) consist of taking the products of the analysis phase and designing all of the relevant portions of the system, the inputs and outputs, and the boundaries of the system (Avison and Fitzgerald 1998). While it was understood that the design & requirements phases might be somewhat iterative, it was accepted that after this phase, the cost of changes to the system based upon changing requirements would increase exponentially (see Figure 2.1). This led to the establishment of "requirements freezes" and sign offs from customers on requirements at this point in the process.

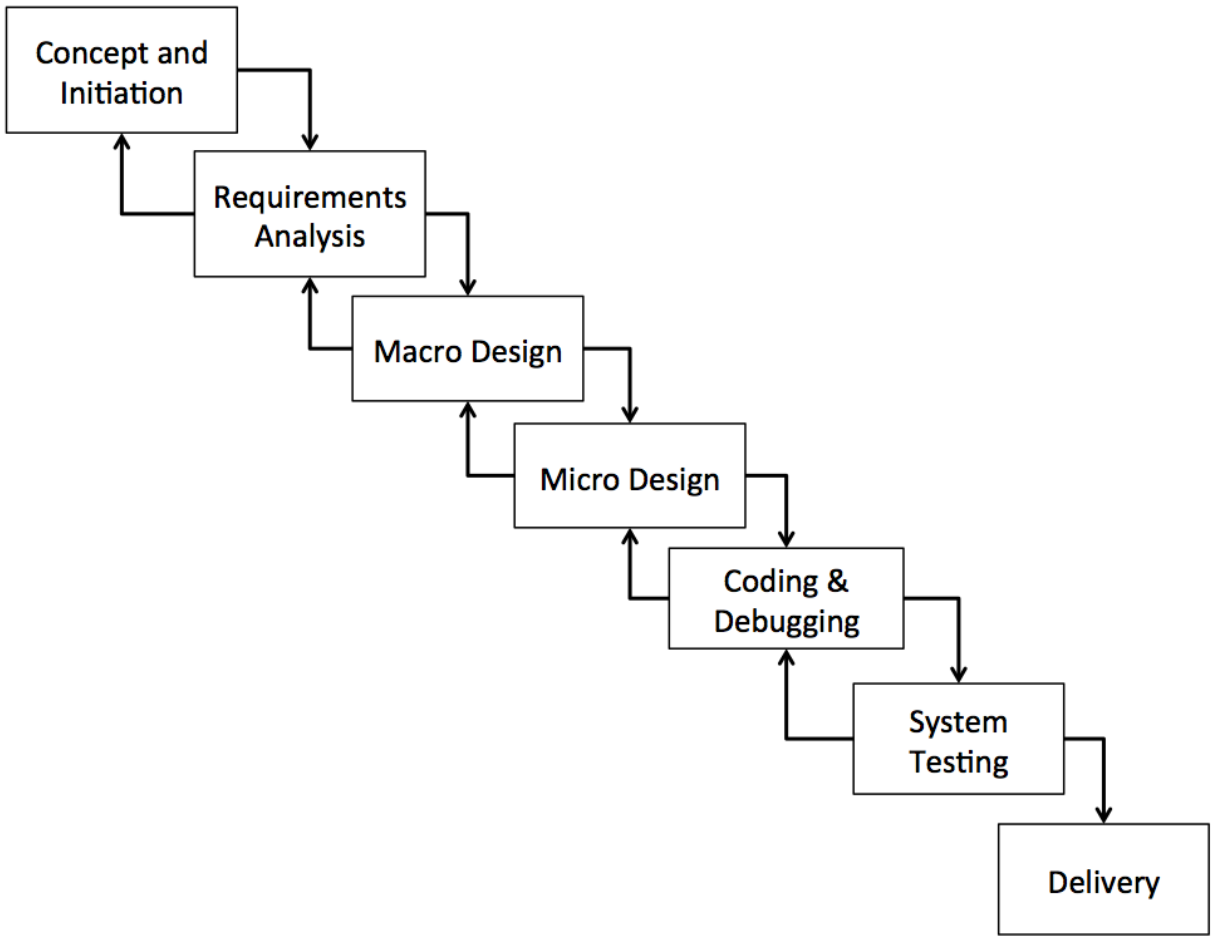

Figure 2.1. The "Waterfall" Method

Once the design stage(s) are completed, the *Coding and Debugging* stage proceeds. During this stage, the development team builds the system, performs unit testing and debugging, and integration testing of the system. Finally, the completed system is delivered for testing in the *System Testing* stage.

The structured method had definite strengths over the previous, ad-hoc model of development. First, it established formal requirements and design procedures. This made for better code quality, and higher probability of acceptance by users. Second, it recognized explicitly the need for formal phases of testing and ongoing maintenance (Boehm 1988).

The method relied on the creation of full documentation of requirements at the very early stages of the project. As Boehm articulates, this is very effective for some classes of software, but not for complex and interactive end-user focused systems, which describes the majority of software projects undertaken (Boehm 1988). These limitations led to the continued evolution of development methodologies.

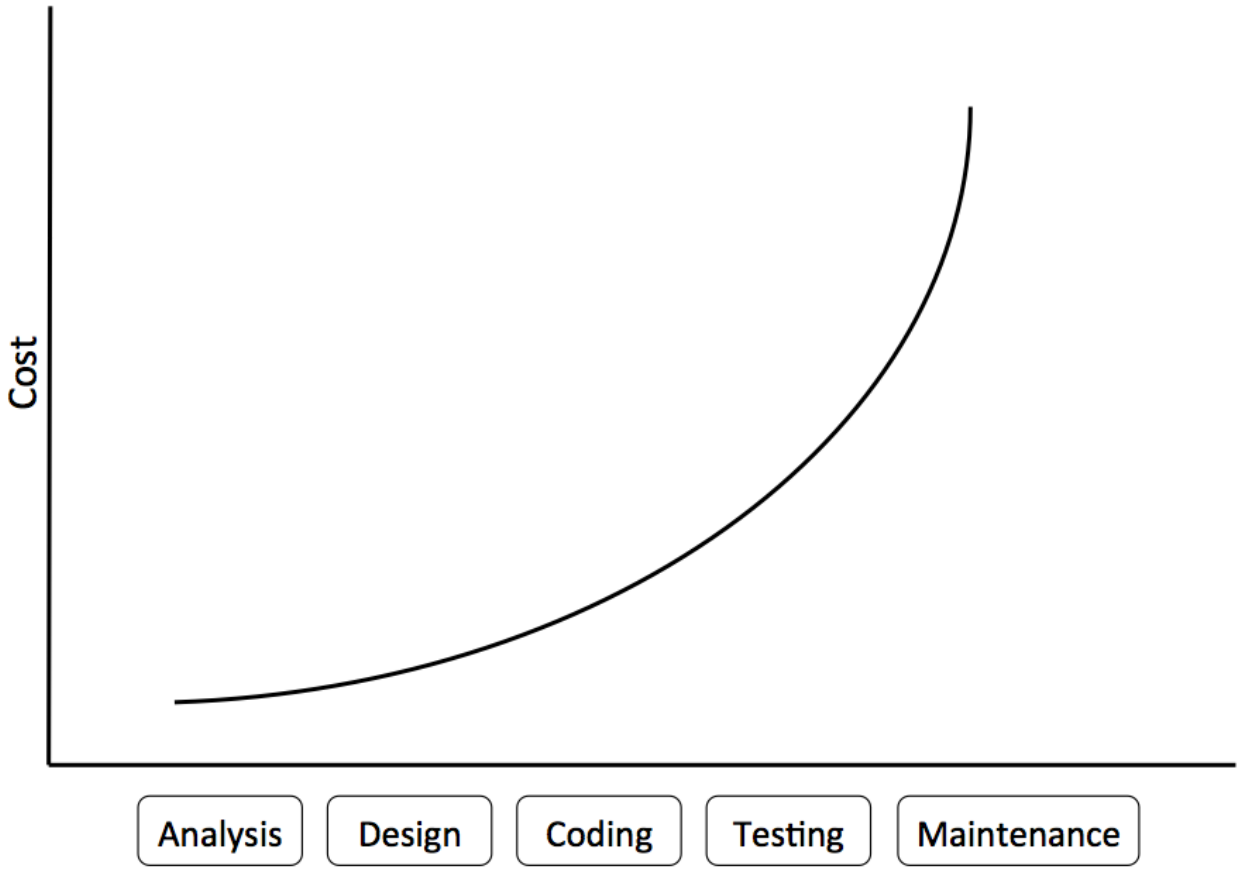

Figure 2.2. The Cost Curve of Change Under The Waterfall Method

The philosophy of the waterfall method assumes the “facts” about the system can be elicited up front with relative accuracy and certainty. This implies that the organization is generally in equilibrium, and that the environment will create little change of the requirements of the system during development. Second, the method assumes that the software development staff can build software from the documented descriptions of analysts. Third, the method assumes that the cost of change (Figure 2.2) becomes significantly higher as time passes, and that changes later in the project should be avoided, hence the focus on early definition and codification of requirements. Finally, the method assumes a linear delivery model, with successive completion of each phase, although later views of the waterfall model allowed for feedback loops between adjacent phases (Boehm 1988). Because of this, the pure waterfall method is

not good at handling changing or uncertain requirements (Boehm 1988). While projects often spent up to 40% of their budgets on requirements elicitation, analysis, and design, this did not prevent changing requirements, which often lead to costly system modifications. Programmers believed that the users continually changed their minds, users believed that the programmers did not give them an accurate understanding of what the system would deliver, and analysts were accused of poorly eliciting requirements (Hirschheim et al. 1995).

Additionally, the timescale of project delivery under the waterfall method is usually months, and often, years. Projects in general have a tendency to escalate due to the need for managers to justify their previous decisions, the psychological impact of sunk costs, and the desire to avoid the negative professional impacts to the project team members (Keil et al. 2000). Because the traditional waterfall method delivers most of the project value at the end of the project causes there to be a long delay before value is delivered, and before there is the ability to measure the progress and success of the project, leading to more pressure to escalate to deliver value from the project.

As can be seen from Figure 2.2 and the description, delivery is conceived of as an event that occurs usually once, usually at the end of a project. This philosophy has dominated the practice of software development. However, the delivery philosophy of agile methodologies is significantly different.

Agile Methodologies – Continuous Delivery

In contrast to the practice of the waterfall method, agile methodologies claim a philosophy to deliver value early iteratively throughout a project. While the assumption of waterfall methodologies was that requirements could be elicited accurately up front

as a “definable” process, agile methodologies approach software development as an “empirical” process. Schwaber (Schwaber 1996) describes these two process types:

> *“If a process can be fully defined, with all things known about it so that it can be designed and run repeatably [sic] with predictable results, it is known as a defined process, and it can be subjected to automation. If all things about a process aren't fully known-only what generally happens when you mix these inputs and what to measure and control to get the desired output-these are called empirical processes.…a defined process is predictable; it performs the same every time. An empirical process requires close watching and control, with frequent intervention. It is chaotic and unrepeatable, requiring constant measurement and control through intelligent monitoring.” (Schwaber 1996)*

Hence, agile methodologies are based on the philosophical assumption that a software method must control an uncertain and ambiguous process. However, neither the practitioner nor research literature has universally adopted this viewpoint. Table 2.4 provides several definitions of agility that have emerged from the practitioner and academic literature. While not completely aligned with each other, there are multiple concepts that are common to all of them.

First, these definitions assume the presence of an interaction between the software development team and its organization (environment). Furthermore, these definitions stress the continuous nature of this interaction. Second is the notion of being able to sense, to learn from, and to respond to change. Third is the concept that through this sensing and responding, swift adjustments that increase customer value can be achieved. This customer value can be achieved via the early delivery of functionality.

Additionally, both Highsmith and Conboy refer to the agile development team possessing the ability to “create” change as well as respond to change. By using this terminology, they indicate that the product of the work of the team – the finished software – provides a cue to the environment. These cues are not available in the

traditional development structure, as working code is rarely delivered and deployed before the end of the project. However, in agile development, the regular delivery of working software is a cue that assists the organization in its ability to understand how well the system will fit into its work processes. Because of this, the system that emerges from the project itself becomes an agent of change (DeSanctis and Poole 1994; Orlikowski 1992).

The definitions of “agility” provided in Table 2.3 are all consistent in their recognition of the need of a method to be structured in a way to receive and respond to environment changes. This indicates the wide recognition of development as being an empirical, ambiguous process. However, while the general philosophy of agile is consistent across agile methodologies, the specifics of the methodologies exhibit significant variance. The next section describes the similarities and differences between the agile methodologies.

Definable vs. Empirical Processes

The ability to forecast the outcome of the development process is at the core of the differences between the philosophies of the traditional and agile methodologies. These methodological perspectives hold conflicting views of the “definability” of the systems development process. If a process is “definable”, its inputs, process tasks, and outputs are understood well enough that, given a specific set of inputs, a predictable and consistent set of outputs can be expected. If a process is “empirical” (or non-definable), control of the process cannot be completed via the planning and standardization of process steps. Rather, an empirical process in controlled by the repeated inspection of its output, associated feedback, and adaptations made based

upon the results of the inspection, which leads to regularity of process (Ogunnaike and Ray 1994). Without the necessity to manage the non-routine, adaptation is unnecessary.

In a group process situation, there are several kinds of inputs. First the resources provided to the team, and the skills of the team members at the beginning of the process can be considered inputs. Additionally, the requirements, or the understanding of the nature and properties of the group's goal are also inputs to the process. The steps necessary to transform the inputs to the process into the goal are the process. Finally, the final product of the group, whether it is a decision, a document, or a software system is the output.

Table 2.3. Selected Definitions of "Agile/Agility"

Source	Definition
IBM Rational (Ambler 2009)	"Agile is the use of continuous stakeholder feedback to produce high-quality consumable code through user stories (or use cases) and a series of short time- boxed iterations."
(Lyytinen and Rose 2006)	"In the context of information system development (development), agility can be defined as an development organization's ability to sense and respond swiftly to technical changes and new business opportunities."
(Conboy 2009)	"The continual readiness of an development method to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value (economy, quality, and simplicity), through its collective components and relationships with its environment."
(Highsmith 2002)	"Agility is the ability to both create and respond to change in order to profit in a turbulent business environment."

When all the components of this process – inputs, process, and output – are well understood, unambiguous, and stable, the process is considered to be “definable” and, therefore, predictable. Most traditional, plan-based software development methodologies are built on the concept of being able to, at the beginning of a project, describe the process needed to complete the project including necessary tasks, their sequence, and inputs and outputs. This assumes a definable process, with steps that can be predicted. This quest for predictability leads to methodologies such as IBM’s Worldwide Project Management Method that defines thousands of potential development process tasks. Even so, as Schwaber (2002) points out, “for the defined control mechanisms to work, these methodologies must define each process with enough accuracy that the resultant noise does not interfere with its repeatability, or the predictability of the outcome.” In short, if a process is truly repeatable, its steps should be defined to such an extent that a team could follow the process like a recipe and, without variation, deliver the anticipated outputs.

In contrast, agile methodologies either explicitly or implicitly adopt the philosophy that the development process is empirical. For this reason, most have adopted the “iterative” development approach discussed above, which attempts to balance the needs for flexibility and adaptation with the need for stability for execution. While agile teams generally develop plans for delivery, they do so in short bursts (iterations), which allow them to commit to delivery, while leaving room to adapt repeatedly based upon feedback. These differences in assumptions regarding definability lead to significantly different planning, management and control (Eisenhardt and Tabrizi 1995).

Agile Development – a Description

Agile Development methodologies are, based on the concept of iterative delivery. Iterations consist of short, time-boxed deadlines, usually no longer than 4-6 weeks (Highsmith 2002; Schwaber 1996; Schwaber and Beedle 2002). Prototyping and other evolutionary methodologies emerged before agile methodologies. However, agile development is differentiated in that in most cases each iteration includes all or most of the steps of the waterfall method. This means that the code necessary to deliver the requirements selected for the iteration is designed, built, tested, and *delivered* (Beck and Andres 2004; Highsmith 2002; Schwaber and Beedle 2002). This iterative delivery loop is illustrated in Figure 2.3.

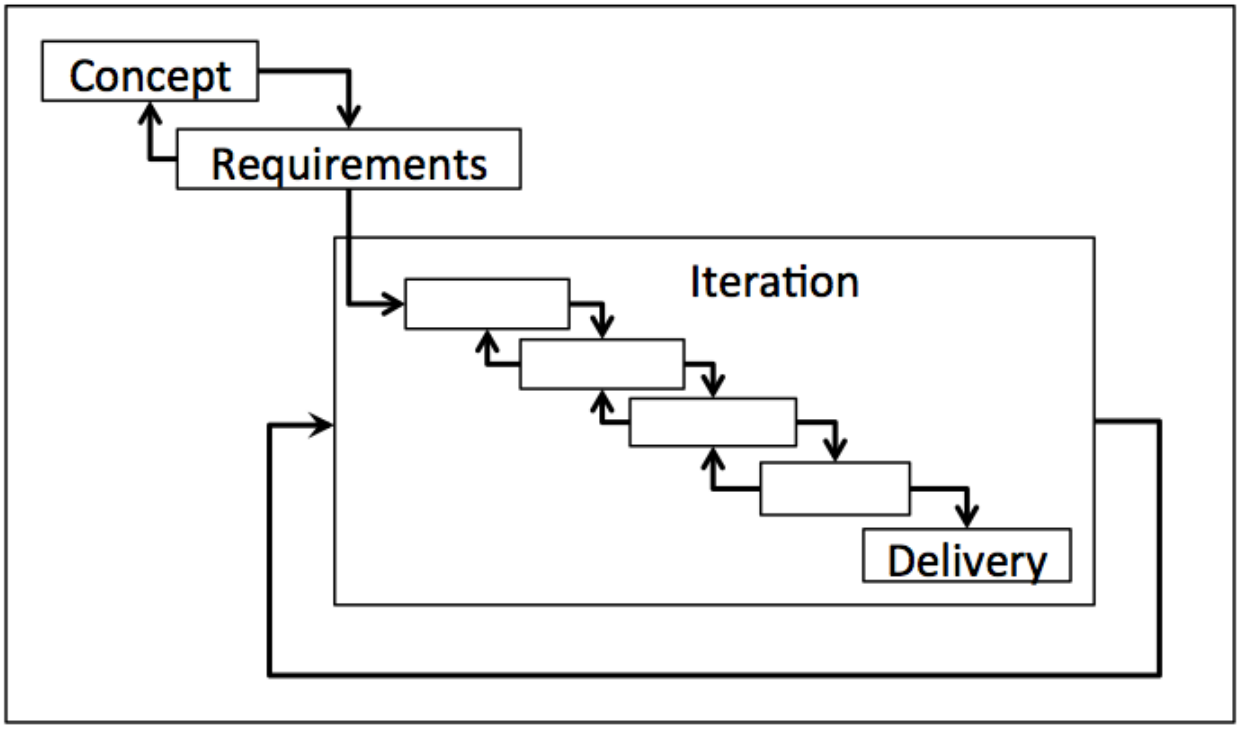

Figure 2.3. Agile Method Feedback Loop

In this delivery model, high-level concept and requirements lists may be developed for the entire system up front. However, only those requirements agreed

upon for the current iteration need to be completed in order for the iteration to begin. Once the iteration begins, the features and associated requirements to be delivered in the iteration are frozen. The team then completes design, development, testing and delivery for all features for the iteration. While this does not necessarily mean that the code is deployed to production, the understanding that the code should be deployable at the end of any iteration, at the discretion of the user is explicit in several methodologies, including Extreme Programming and Scrum (Beck 1999; Highsmith 2002; Schwaber and Beedle 2002).

Teams require some stability in order to take action (Argyris and Schoen 1978; Gersick and Hackman 1990; Okhuysen 2001). By organizing in the fashion described above, the agile development team attempts to create a balance between stability and flexibility. The team provides flexibility and adaptability by receiving feedback cues from the environment at least at the end of each iteration. These feedback cues are of high quality because, rather than being based upon representations of the system as would occur during the analysis and design phase of a waterfall project, they are based upon a review of the working software (Lave 1991). This working software is a far richer medium for users to evaluate whether the system that is being built is suitable for the task.

Additionally, because the team chooses a small portion of the system's features to build during an iteration and then freezes the requirements for those features during the execution of the iteration, they provide stability for action. Additionally, this in-process requirements freeze provides the efficiency of buffering the team from the environment, and allowing the team to execute the process as efficiently as possible

(Thompson 2003). In a real sense, this process enables the team to *behave* as if the software development process is definable during the iteration, while embedding these episodes of definability within a structure of empirical reflection and adaptation.

To illustrate an episode of definability, in each iteration, a small subset of the system features is chosen to be addressed. These features are usually the most valuable (as defined by the user) remaining to be completed (Schwaber and Beedle 2002). During the iteration, while these features are being completed, requirements for other parts of the system can be changed. Changes in requirements for the portion of the system under development are postponed until a later iteration. The timeframe of the iteration is only a maximum of four to six weeks (Schwaber and Beedle 2002) (Beck 1999), and often substantially less (Highsmith 2002). Therefore, very few requirements that emerge during a cycle will rise to the level of importance as to merit an exception to this rule.

However, at the end of this episode of definability, the team intentionally emerges from behind the buffer and displays the results of the iteration. This review of the functional code allows for a determination to be made regarding the fit of the developed software to the requirements of the environment as it stands at the end of the iteration. Incongruities spawn new feature requests, and are prioritized as are any other features.

After each iteration is completed, all agile methodologies prescribe a retrospective assessment. During this time, the team reflects upon the process performed during the priori iteration, and the source of any incongruities between requirements and delivered software. Additionally, they may adapt by implementing process changes speculated to generate higher performance in the next iteration.

Finally, agile development practitioners propose that the time-boxed iteration allows the team to eliminate several traditional software development artifacts. While traditional methodologies focus on up front planning and documentation, the iterative time-box approach allows for learning during the project. Rather than working to provide a full enumeration of requirements as a control mechanism up front, control is established via the time-box (Highsmith 2002). Additionally, because the system is delivered each iteration, the system itself acts as documentation, removing the need to build and maintain abstract documentation artifacts (Consortium 2002-2011).

Agile practitioners (Beck 1999; Highsmith 2002) claim that this structure of adaptive iterations, enabled by agile method engineering and process practices, provides a lower cost curve of change. Because the system is built in small steps, delivered in short durations, the agile methodologists claim that the cost of change curve that impacted waterfall projects to a far higher degree in later stages can be significantly flattened (See Figure 2.4).

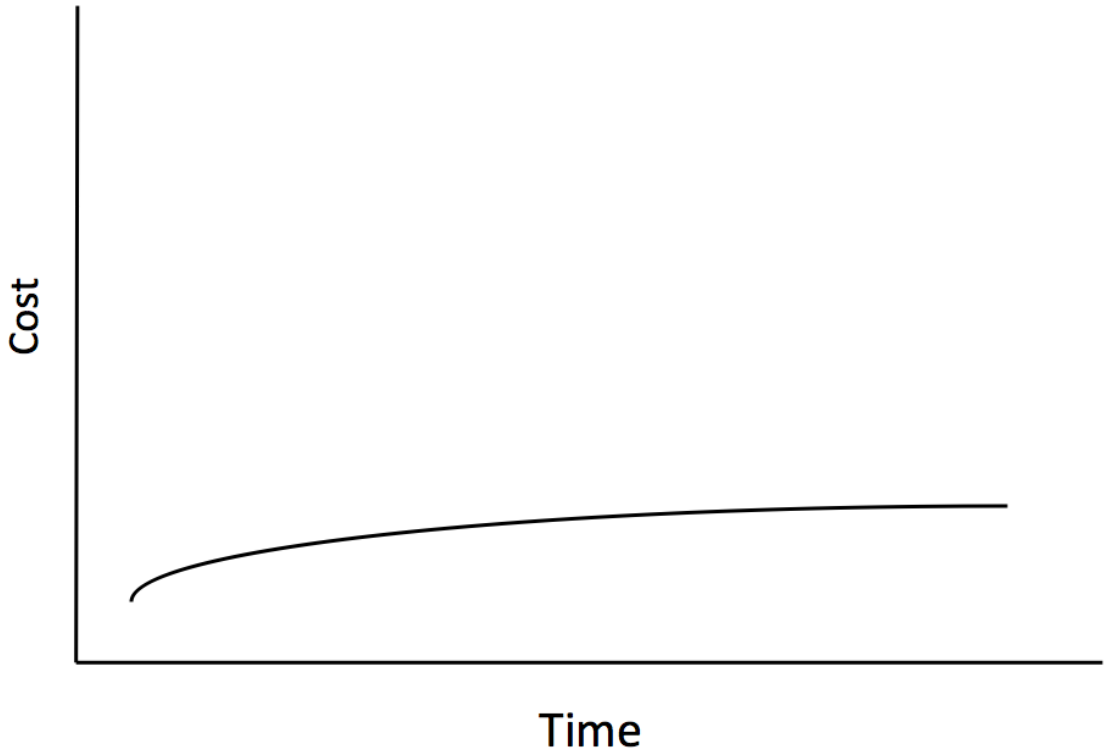

Figure 2.4. The Cost of Change Curve Under Agile Method (Beck 1999)

The Practitioner Literature on Agile Methodologies

One of the first formal articulations of the agile philosophy came with the publication of the *Manifesto for Agile Software Development*, a document outlining so-called "agile" development as an alternative mode of software production. The signatories of the manifesto had already been practicing various "agile" methodologies for several years (Beck 1999; Schwaber 1996). As such, the *Manifesto* is a statement of shared philosophical beliefs held by the proponents of common agile methodologies.

The Agile Manifesto is structured into a value statement and twelve foundational principles. The value statement is reproduced here:

> *"We are uncovering better ways of developing software by doing it and helping others do it. We value:*
>
> - *Individuals and interactions over processes and tools.*

- *Working software over comprehensive documentation.*
- *Customer collaboration over contract negotiation.*
- *Responding to change over following a plan.*

That is, while there is value in the items on the right, we value the items on the left more." (Beck et al. 2001)

As explained by Martin Fowler and James Highsmith in their exposition of the Agile Manifesto (Fowler and Highsmith 2001), this purpose statement is a description of the foundational philosophy of agile development. In each of the value statements, two valued items are contrasted. Fowler and Highsmith are clear, both items on each bullet point are valued, but the item on the left is of more value. Fowler & Highsmith's commentary throughout their article indicates that the value statements drive decisions that must be made when change arises:

"Contract negotiation, whether through an internal project charter or external legal contract, isn't a bad practice, just an insufficient one. Contracts and project charters may provide some boundary conditions within which the parties can work, but only through ongoing collaboration can a development team hope to understand and deliver what the client wants.

No one can argue that following a plan is a good idea—right? Well, yes and no. In the turbulent world of business and technology, scrupulously following a plan can have dire consequences, even if it's executed faithfully. However carefully a plan is crafted, it becomes dangerous if it blinds you to change. We've examined plenty of successful projects and few, if any, delivered what was planned in the beginning, yet they succeeded because the development team was agile enough to respond again and again to external changes." (Fowler and Highsmith 2001)

Therefore, according to the agile manifesto, an agile philosophy is one in which individuals are valued, working software is the key indicator of progress, customer interaction is rich and continuous, and change is an accepted part of the development

process. The agile manifesto further defines its philosophy via the 12 Principles of Agile Development (see Table 2.4).

The principles can be loosely categorized into five categories. (1) **Iterative Delivery** – all agile methodologies stress this principle, namely the importance of delivering incrementally. (2) **Product Focus** – the product – working software – should be the measure of success and project value delivery. (3) **Self-directed team** – the team should have the power to organize itself, and should be responsible for maintaining its process. (4) – **Feedback** – teams should have an involved user, and should communicate regularly. (5) **Responsiveness** – The team should receive feedback, but also be willing to act in accordance with the information contained in the feedback. At a high level, these five categories of principles relate to both the philosophy of agile methodologies, and the actions or practices of agile methodologies.

Table 2.4. The 12 Principles of the Agile Manifesto

Principle	Clarification (Fowler and Highsmith 2001)
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	"…we need to understand that customers don't care about documents, UML diagrams or legacy integration. Customers care about whether or not you're delivering working software to them every development cycle—some piece of business functionality that proves to them that the evolving software application serves their business needs."
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	"Rather than resist change, the agile approach strives to accommodate it as easily and efficiently as possible, while maintaining an awareness of its consequences. Although most people agree that feedback is important, they often ignore the fact that the result of accepted feedback is change. Agile methodologies harness this result, because their proponents understand that facilitating change is more effective than attempting to prevent it."
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale.	"For many years, process gurus have been telling everyone to use an incremental or iterative style of software development, with multiple deliveries of ever-growing functionality. While the practice has grown in use, it's still not predominant; however, it's essential for agile projects. Furthermore, we push hard to reduce delivery cycle time."
Business people and developers work together daily throughout the project.	"Many folks want to buy software the way they buy a car. They have a list of features in mind, they negotiate a price, and they pay for what they asked for. This simple buying model is appealing, but for most software projects, it doesn't work."

Table 2.4. (cont'd)

Principle	Clarification (Fowler and Highsmith 2001)
Build projects around motivated individuals, give them the environment and support they need and trust them to get the job done.	"Deploy all the tools, technologies and processes you like, even our agile processes, but in the end, it's people who make the difference between success and failure."
The most efficient and effective method of conveying information with and within a development team is face-to-face conversation.	"Inevitably, when discussing agile methodologies, the topic of documentation arises. Our opponents appear apoplectic at times, deriding our "lack" of documentation. It's enough to make us scream, "the issue is not documentation—the issue is understanding!" Yes, physical documentation has heft and substance, but the real measure of success is abstract: Will the people involved gain the understanding they need?"
Working software is the primary measure of progress.	Too often, we've seen project teams who don't realize they're in trouble until a short time before delivery. They did the requirements on time, the design on time, maybe even the code on time, but testing and integration took much longer than they thought.
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	Agility relies upon people who are alert and creative, and can maintain that alertness and creativity for the full length of a software development project. Sustainable development means finding a working pace (40 or so hours a week) that the team can sustain over time and remain healthy.
Continuous attention to technical excellence and good design enhances agility.	When many people look at agile development, they see reminders of the "quick and dirty" RAD (Rapid Application Development) efforts of the last decade. But, while agile development is similar to RAD in terms of speed and flexibility, there's a big difference when it comes to technical cleanliness. Agile approaches emphasize quality of design, because design quality is essential to maintaining agility.

Table 2.4. (cont'd)

Principle	Clarification (Fowler and Highsmith 2001)
Simplicity--the art of maximizing the amount of work not done-- is essential.	It's easier to add something to a process that's too simple than it is to take something away from a process that's too complicated. Hence, there's a strong taste of minimalism in all the agile methods. Include only what everybody needs rather than what anybody needs, to make it easier for teams to add something that addresses their own particular needs.
The best architectures, requirements, and designs emerge from self- organizing teams.	that the best designs (architectures, requirements) emerge from iterative development and use rather than from early plans. The second point of the principle is that emergent properties (emergence, a key property of complex systems, roughly translates to innovation and creativity in human organizations) are best generated from self-organizing teams in which the interactions are high and the process rules are few.
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	Agile methods are not something you pick and follow slavishly. You may start with one of these processes, but we all recognize that we can't come up with the right process for every situation. So any agile team must refine and reflect as it goes along, constantly improving its practices in its local circumstances.

A full comparison of major methodologies is beyond the scope of this dissertation[2]. For a full Table 2.5 provides a short summary of the practices encouraged by each method. In brief, the majority of the methodologies focus on coordination and communication practices, rather than development and coding practices. The significant exception to this statement is Extreme Programming (XP). As illustrated in Table 2.5, most methodologies focus on coordination and communication practices, indicating the shared focus on developing the ability to both receive and

[2] For an excellent exposition of the various agile methods, see Highsmith (2002)

process feedback. Because of the various methodologies' unique evolution, they define differing enabling processes and practices that are specific to their focuses. This is why some methodologies have developed enabling practices that are in conflict. The concept of enabling processes in agile methodologies will be discussed more fully below.

Finally, the importance of the "tools" of agile development – source code control, continuous integration software, and automated unit testing has been stressed since the beginning of the agile movement (Beck 1999; Highsmith 2002). The practitioners stress that the agile practices are enabled by a complex system of people, processes and technology, and that these entities reinforce and enable one another.

In summary, the philosophy of agile development stresses the value of completed code that meets customers' needs. In order to delver software based upon that philosophy, agile methodologies each propose a set of direct and supporting practices, processes, and technologies that they claim will enable higher levels of success in software delivery. However, we argue that the defining practices of agile methodologies, and those most likely to directly impact project success, are those practices that enable feedback.

We now turn to the theoretical lenses through which we will build my theoretical model of the impacts of agile software development on project success. My analysis relies on three theoretical perspectives for grounding. First, uncertainty theory describes when agile methodologies are most likely to be impactful. Second, dynamic capability theory describes how organizations develop the ability to sense and respond to environmental cues under uncertainty. Finally, team adaptability theory describes a

complex network of team norms, practices and processes that enhance performance under team-level uncertainty.

Table 2.5. Defined Practices of Major Agile Methodologies

Practice	XP (Beck 99)	XP (Beck 04)	SCRUM	FDD	Crystal Clear	ASD
Pair Programming	Yes	Yes				
Collective Code Ownership	Yes	Yes				
Continuous Refactoring	Yes					
Continuous Integration	Yes	Yes	Yes			
Unit Testing	Yes	Yes	Yes		Yes	
Coding Standards	Yes					
Planning Game	Yes		Yes			
Small Releases	Yes		Yes	Yes	Yes	Yes
Metaphor	Yes					
Simple Design	Yes					
Sustainable Pace	Yes					
On-site customer	Yes					
Daily Short Meeting	Yes		Yes			
Iteration Retrospective meeting			Yes		Yes	Yes
Formal Iteration Review (with customer)			Yes		Yes	
Sit Together		Yes			Yes	
Informative Workspace		Yes				
Weekly Cycle		Yes				
Slack		Yes				
Incremental Design		Yes				
Whole Team		Yes				
Energized Work		Yes				
Stories	Yes	Yes		Yes	Yes	
Quarterly Cycle		Yes				
Ten-minute build		Yes				
Test-first programming		Yes				
Real Customer Involvement		Yes			Yes	
Team Continuity		Yes				
Root-cause analysis		Yes				
Code & Tests		Yes				

Table 2.5. (cont'd)

Practice	XP (Beck 99)	XP (Beck 04)	SCRUM	FDD	Crystal Clear	ASD
Daily Deployment		Yes				
Pay-per-use		Yes				
Shrinking Teams		Yes				
Single Code Base		Yes				
Negotiated Scope Contract		Yes				
Develop an overall model				Yes	Yes	
Build a Features List			Yes	Yes	Yes	
Plan by Feature				Yes		
Design by Feature				Yes		
Build by Feature			Yes	Yes		
Chief Programmer				Yes		
Class Owner				Yes		
Feature Teams				Yes		
Project Documentation			Yes	Yes	Yes	
Develop a Mission Statement						Yes
Time boxed cycles						Yes
Burndown			Yes			

Uncertainty: Complexity and Change

As noted above, uncertainty has been characterized as the fundamental problem that faces any organization (Thompson 2003). Uncertainty means "the degree to which future states of the world cannot be anticipated and accurately predicted" (Pfeffer and Salancik 2003, p. 67). In the organization theory literature, uncertainty has been posited to arise from a number of sources. The complexity and dynamism of an environment are directly related to the inability to anticipate and accurately predict future states of the environment and organization.

Complexity of various forms has been shown to impact the success of software development projects (Xia and Lee 2005). Software projects are often highly complex by nature, because they must manage both the technical complexity of the project environment, and the organizational complexity of the environment in which the project

takes place (Baccarini 1996; Brooks 1995; Kirsch 1996; Kirsch et al. 2002; Xia and Lee 2005). IS Project uncertainty therefore is based upon both the structural and technical complexity of the project, as well as the complexity that comes from organizational and requirements dynamism (Williams 1999; Xia and Lee 2005).

Structural complexity refers to the complexity of the organizational environment in which the project takes place. Project teams are often temporary in nature, and utilize members of multiple departments and other functions. Structural complexity refers specifically to the size of the project team, the number of departments for which the team is attempting to deliver a solution, the number of departments to which the team members report, and the geographic distribution of the project team.

As the number of departments that are represented in the project, whether as stakeholders providing requirements for the team, or as members of the team increases, leads to an increase in the difficulty of establishing well-defined and stable project goals. The lack of these defined and stable goals has been posited as a key driver of project complexity and reduction of project success (Turner and Cochrane 1993).

Project teams require regular feedback to ensure that they are progressing toward completion. This feedback is obtained through formal status reporting, serendipitous discussions, ambient awareness within the team area, and other forms of direct contact. Structural complexity increases when the mechanisms that occur naturally in team environments are impeded due to the lack of direct communication.

Finally, structural complexity is associated with the criticality of the project and its associated deliverables. Traditionally, project tasks are either defined as “critical” or “not

critical". Critical tasks are those that, if delayed, cause delay in the entire project (Kuchta 2001; PMBOK 2000). However whole projects are recognized as having levels of criticality. When projects are critical, their potential positive and/or negative impact to the organization is greater. However, in project portfolio management and software method literatures, the criticality of a project is generally measured in terms of its potential negative impact.

Technical complexity arises from the combination of scope and structure of the project system, and the ambiguity regarding the use of various technologies to complete the project (Shenhar and Dvir 1996).

The dynamic nature of the systems development environment is reflected in the extent to which the project team must respond to changing requirements throughout the duration of the project. Changing requirements are caused by ambiguity in initial requirements definition, as well as actual volatility in the system requirements (McKeen et al. 1994; Meyer and Curley 1991; Ribbers and Schoo 2002).

Uncertainty in systems development environments is defined as the extent to which the development team cannot accurately identify the necessary inputs, tasks, and sequencing necessary to ensure the desired outcome of the systems development process, due to the impacts of the complexity of the environment and deliverable, and the dynamic nature of the environment. In an uncertain environment, organizations must adapt, rather than rely on pre-defined plans to be successful. The next section contains a discussion of team adaptability theory, which describes mechanisms through which teams may react to uncertain environments.

Team Adaptability & Organizational Learning

A software development team is a specialized form of small group. Small group researchers in social psychology and organizational behavior have investigated the interaction between teams and their environments at length. Groups originally were viewed as a unit with firm boundaries, conceived as simply the sum of the properties of the individual team members. Teams were conceived of as black boxes that received inputs, processed them, and produced output (McGrath 1964). Group performance research focused on what characteristics of team members and team processes might encourage or hinder efficiency, and group success was thought to be indicated by this efficiency (Ancona 1990; Ancona and Caldwell 1992). However, more recently, researchers have focused less on the how of team process and more on why some teams perform at high levels while others don't (Ilgen et al. 2005). This literature has moved beyond the borders of the team and has focused on the complex interplay between the team and its environment (Burke et al. 2006; Kozlowski et al. 1999; McGrath et al. 2000).

Team adaptation refers to the process through which teams develop a shared vision of their goals, enact plans, norms, processes and procedures to achieve those goals, and proceed to execute those plans to achieve the goals. As the team executes the plans, it receives feedback from the environment and, if a "salient cue" is identified, the team adapts to the cue, and re-formulates its plan in reaction to the cue (Burke et al. 2006). "Team adaptation is manifested in the innovation of new or modification of existing structures, capacities, and/or behavioral or cognitive goal-directed actions" (Burke et al. 2006). Adaptive teams are indicated by their ability to sense barriers to

performance, and to modify their practices and goals to effective manage these barriers. This adaptation can take the form of barrier removal or barrier avoidance. (Tesluk and Mathieu 1999). Whether emerging from organization, environmental or technical forces, adaptive teams develop the ability to first sense the barrier (or potential barrier), and then take action to remove the barrier's impact on team performance. In method environments, these barriers may stem from the policies of the customer, the impact of external system constraints, such as enterprise architecture requirements, or the ability for the technology platform to accommodate changes.

Teams are required to adapt to non-routine situations. Agile development views software development as unpredictable, non-routine and novel. When responding to novel conditions, teams whose members can recognize that the environment had changed were shown to be more adaptable (Waller 1999). Additionally, teams that interrupted their processes to "stop and think" about the process and its performance were shown to be able to recognize the need to respond to environmental change more quickly (Okhuysen and Waller 2002).

Team adaptability theory proposes several team processes and emergent states that impact their adaptation capability. Burke et al. (2006) propose an integrated process model of adaptation that bears strong resemblance to the agile methodologies in practice. They describe an iterative, feedback intensive learning process characterized by receiving repeated cues from the environment, ascribing meaning to and learning from those cues, and adapting team behavior based upon the new learning.

Team adaptation is a cyclical process. At each stage of the cycle, teams either receive or react to cues from the environment. These stages lead to the creation of, and modification of shared mental models. Teams build situational awareness and a plan to execute the team tasks in Phase 2. As the plan is executed, the teams coordinate their activities via mutual monitoring, and coordination, which lead to learning and feedback. Agile development requires the ability to sense and respond to change (Conboy 2009; Lyytinen and Rose 2006) Therefore agile development methodologies propose practices that map closely to the constructs developed by Burke et al. (2006). These characteristics are described below.

Cue Recognition

The presence of cues is not equivalent to recognition and action due to those cues. Gersick & Hackman (Gersick and Hackman 1990) found that when the responses to pre-flight checklists became habituated, even the presence of cues that would require a modification to the pre-flight checklist procedure did not cause a change in action. In this case, the flight crew (in other words the team) had become so familiar with the responses acceptable in one situation (flying in warm, southern climates), that when the situation changed (taking off in an ice storm), the crew could not recognize the need for the process to change. So, the fact that environmental cues exist does not indicate the ability for the team to assess that cue as being relevant to taking action.

A team must be able to identify which cues are triggers for adaptation. Louis and Sutton (Louis and Sutton 1991) describe the cognitive “shifting of gears” which is required to pull thinking from the automatic mode to the conscious mode. They identify several situations that can cause teams to recognize cues in either a planned or

unplanned manner. When teams experience unusual or novel events, an incongruity between expectation and experience, or the specific initiative to draw attention to cues, and opportunity for cue identification arises. Okhuysen (2001) found that groups who were given a mechanism for formally diagnosing process discrepancies generated more work interruptions were able to adapt

Since Agile methodologies are based upon the philosophy of delivering in small pieces and in short cycles called iterations, they provide the opportunity for repeatedly receiving cues from users as to whether the shared mental model begin developed regarding the problem and the solution is correct. Further, because planning occurs at the iteration level, external environmental changes can be detected and incorporated into the plan for the next iteration.

Cues are not limited to information crossing the team boundary, but can also be associated with temporal structures within a team's execution plan. Gersick (1988) found that teams reevaluate and revise their execution plan based upon the cue of the project schedule mid-point. Additionally, she found that the mid-point of a project acted as a catalyst for adaptation. Interestingly, this study showed that the length of time between the deadlines did not matter; rather the mid-point of the schedule was the catalyst for change, not the duration of time passing. This suggests that by taking advantage of this seemingly built-in trigger for adaptation moves, teams who plan shorter duration milestone deadlines should receive more cues to adapt than those who schedule longer duration milestones.

Team Situational Awareness

"Team situation awareness refers to a shared understanding of the current situation at a given point in time" (Salas et al. 1995) Situational awareness is the product of interaction of individual situational awareness of the team members, and the degree of the team members' shared understanding of the environment and task, in other words, the team's shared mental models (Salas et al. 2001). Endsley's model of situational awareness (Endsley 1995) describes the existence of three levels of increasing complexity of situational awareness. Level one awareness describes the ability to simply perceive cues from the environment. Level two awareness exists when a team is able to comprehend the relevance of its environmental cues, as to their relationship to the current task. Finally, by developing the capacity to evaluate the relevant environmental cues and extrapolate likely outcomes of the current team task, based upon the new cues, a team attains level three awareness.

Developing and maintaining team situational awareness is a focus of several agile practices. As will be discussed below, Agile development methodologies stress the importance of two situational awareness-building processes. First, the team receives feedback about the fit of the output of the team task (the system) each iteration. Based upon this feedback (cue), the team evaluates both the output of the process and the process itself. This process provides the team the ability to receive and process cues, and to evaluate and forecast performance. Second, because most agile methodologies call for the establishment of a daily short review meeting, situational changes are disseminated to the team repeatedly throughout the iteration itself, allowing for the

perception and reaction to cues more often, and more quickly than other methodologies that call for longer periodic status update reporting.

Mutual Monitoring

Mutual performance monitoring is a process through which team members observe the actions of their teammates, and attempt to identify mistakes and other performance issues. By making this identification, teams can expeditiously correct performance issues. Mutual performance monitoring implies that team members still can complete their own work while keeping track of the work of other team members and ensuring compliance with existing team procedures. (McIntyre and Salas 1995)

Mutual performance monitoring contributes to a team's ability to adaptively execute a plan in several ways. Marks & Panzer (2004) found that mutual performance monitoring enables the recognition of when team members need assistance, increases coordination within the team, and increases team performance. Mutual performance monitoring provides intra-team adaptive cues, in that it provides a mechanism to alert the team that the team process may be off track (Dickinson and McIntyre 1997). Additionally, mutual performance monitoring helps to build team situational awareness, leading to a greater ability to sense environmental cues (Salas et al. 1995).

Agile development methodologies prescribe mutual monitoring via several practices, including pair programming, chief programmer reviews, code standards and daily stand up meetings.

The process of team adaptation described above is an organizational learning process. As teams build shared mental models in order to understand their tasks, processes, and goals. Based upon this shared understanding, the teams build a plan of

action, and execute the plan. They team establishes the plan based upon their theorized understanding of the goals of the task.

As teams act on their expectations of outcomes, the learning process occurs when there is incongruity between the goals of the team, and the perceived outcomes (Argyris and Schoen 1978). However, there are several sources of potential incongruity between the established goals of the team, and the delivered output of the process. First, the team may have developed an inaccurate view of the goal based upon misunderstanding of cues from the environment. Second, the team's established plan may not deliver the outputs it was expected to. Third, it is possible that the team delivered the output desired, accurately representing the previous cues from the environment, and yet the output is not acceptable, either due to changes in the environment during the execution process, or due to the fact that the source of the cue was incorrect in establishing the team's goal. In short, learning occurs when signals are received that are incongruent with the team's expectations (Pich et al. 2002).

However, for learning to occur within the team, "learning agents' discoveries, inventions and evaluations must be embedded in organizational memory. They must be encoded in the individual images and the shared maps of the organization theory-in-use from which individuals will subsequently act. If this encoding does not occur, individuals will have learned but the organization will not have done so" (Argyris and Schoen 1978).

Adaptive teams reform their shared mental models based upon the detection of incongruity (via situational awareness and cue sensitivity). After detection and reformation, adaptive teams encode their learning into the theory-in-use during each adaptive cycle.

Software Development As New Product Development

In the early history of building software, software creation efforts were often characterized as software engineering. As such, early software development methodologies bear similarity to plan-based engineering project methodologies (Boehm 1988). However, agile practitioners argue that the process of developing software bears a much greater resemblance to new product development efforts (Highsmith 2002). A great deal of literature exists that describes the issues of product development under uncertainty. This literature presents several congruent perspectives about the impact of uncertainty on planning and new product development.

MacCormack & Verganti (2003) investigated 29 internet companies and the impacts of two specific types of uncertainty – platform uncertainty and market uncertainty – on the impacts of the choice of development process on project performance. The authors suggest that various types of uncertainty and other contingencies might lead to different responses by the team in the design of their development process. They look at the concept of flexibility of development process as being a core response to uncertainty.

Bhattacharya, Krishnan & Mahajan (1998) defined a concept of “Real-time definition” of product requirements. They found that in new product development, contingencies drive trade-offs between early and late product specification locking. In highly uncertain environments, either due to marketing uncertainty or the lack of full understanding of customer’s requirements, iterative design with multiple rounds of feedback was found to lead to higher performance.

Agile development teams are intended to rapidly respond with new solutions. Eisenhardt and Tabrizi (1995) compared two competing strategies for making rapid innovations, the compression strategy and the experiential strategy. The assumptions of these two strategies are analogues to the assumptions of plan-based and agile development processes. Compression based strategies are based upon the assumption that the product development process can be defined with a series of pre-defined steps. Because the steps are definable, the compression strategy states that the steps can be performed in an overlapping fashion, by shortening the steps themselves, and by rewarding developers for schedule completion.

On the other hand, experiential strategies are based on the assumption that the product development process is marked by uncertainty in the environment; both requirements-based uncertainty and technology-based uncertainty. They note, “the key to fast product development is, then, rapidly building intuition and flexible options in order to learn quickly about and shift with uncertain environments. At the same time, it is also important to create structure and motivate pace in these settings” (p 91). This suggests that in uncertain environments, building situational awareness and a shared team mental model is a key to both team adaptation in general, and to the speed of task completion.

In experiential environments, the authors found that the number of iterations increased development speed through shared understanding. Additionally they find that product development team is shortened due to early feedback (via testing), due to the fact that it gives developers early feedback regarding failures. Finally, they find that the reduction in time between milestones reduces project development time. Gersick (1988)

found that teams reorganize task efforts and develop urgency to complete assigned tasks at the midpoint of the task schedule, regardless of the length of time schedules.

Eisenhardt & Tabrizi (1995) found that, in general, projects that were more definable could be delivered more quickly using a compression strategy, and that they were not able to be delivered more quickly via an experiential strategy. For uncertain environments, they found the reverse to be true. Importantly, their analysis results showed that when each strategy was applied to each

Similarly MacCormack & Verganti (2003) found that early feedback and adaptation that continued throughout the length of the project was important to project performance. They found that under uncertainty significant differences in investment in architectural design expenditures, early technical feedback and late changes to requirements all predicted performance. MacCormack (2001) also found that rapid and early feedback positively impacts product quality and performance.

As the discussion above illustrates, under conditions of uncertainty, the new product development literature illustrates the impact on project outcomes of greater depth and frequency of feedback.

Adaptability, Agile and Feedback Capabilities

The theories and literature presented above share significant commonality in their definition of necessary conditions for develop team adaptation capabilities. According to Team Adaptability Theory, Organizational Learning Theory, and the new product development literature, the central capability necessary to respond to uncertainty is the ability to obtain feedback.

It was noted above that agile methodologies' internal practices are quite diverse and have diverse focuses. However, what is universally accepted in agile methodologies is the need for continuous feedback. Every agile method except Lean/Kanban argues for an iterative delivery cycle. Lean/Kanban takes the iterative cycle to the extreme, considering each development action as an iteration.

The iterative development cycle is an excellent illustration of team adaptability theory in action. At the beginning of each iteration, the team and stakeholders assess the current situation, including processing feedback from the previous iteration. They evaluate the current needs of the organization, and develop a shared mental model for the environmental status, organizational needs, and the deliverables for the coming iteration. Next, they formulate the plan to deliver the features of the iteration. After this, they execute the iteration, process cues and coordinate while delivering. Finally, they process the results of the iteration and determine necessary modifications to process for the next delivery cycle.

By performing the entire product development lifecycle for the given requirements during every iteration, agile methodologies "freeze" the requirements at the beginning of every cycle. This makes the environment and task conceptually definable, and the team can take advantage of the compression based speed enhancement mechanisms. However, at the end of each iteration, the agile methodologies dictate the use of several feedback mechanisms, including system reviews with the stakeholders and users, and retrospective analyses of the iteration processes. This rapid feedback loop provides the three characteristics of the

experiential process. Iterations are short, are of high number and constitute the repeated tests of the product necessary to guide its speedy development.

In a sense, the agile development team treats the macro process (the project) as an empirical process, and the micro process (the iteration) as definable. Because building a shared understanding and a theory in practice is necessary for action, the agile development team must believe that what they are to act upon immediately is knowable, even if what is to be defined later is not.

This discussion suggests that rapid, early feedback should impact agile performance, due to the experiential nature of agile project management. Additionally, the research of Gersick (1988) suggests that shorter, iterative development milestones should result in more rapid response to change, and faster delivery due to the motivation of pace (Eisenhardt and Tabrizi 1995).

Project Success

A common way to contrast agile development with traditional project methodologies is to label projects as “agile” vs. “plan based”. Traditional project management is based upon the assumption that accurate plans can be created for the entire project, up front. Because of this, project management performance metrics are usually based on the conformance of team process and outcomes to the plan. However, in the traditional project management field, measuring project performance by the “Iron Triangle” (Scope, Time & Budget) has begun to fall out of favor. Rather, a broader definition of project success has been proposed (Atkinson 1999; Phillips et al. 2002). These calls for broader definitions of project success call for the evaluation of project process and outcomes across four categories: (1) Project management metrics, (2) the

quality of the project output, (3) individual benefits, and (4) organizational benefits (Atkinson 1999).

Project Management Metrics

The “Iron Triangle” of project management metrics, Cost, Quality & Scope, has been the traditional measure of project “success”. Consistent with this tradition is the CHAOS report from the Standish Group, one of the most widely disseminated sources of IT project result data, which utilizes only the Iron Triangle metrics to define project success. Standish Group places projects into three categories, Success, Challenged, and Failed, based upon these criteria:

> *“A project is successful if it is completed on time, within budget, and with all features and functions as initially specified. A project is challenged in case it is completed and operational but over budget, over the initial time forecast, and if it offers fewer features and functions than originally specified. A project is impaired or failed if it is canceled at some point during the development cycle” (Eveleens and Verhoef 2009)*

Based upon these criteria, the CHAOS report has shown that project success is different when measured in project management metrics. Table 2.6 shows the Chaos categorized results for selected years of the report.

However, when one adopts the view of IT projects as empirical processes, it is predictable that a report that relies wholly on project management metrics would show significant project failure rates. Project Management metrics are based upon the ability to forecast accurately, at the beginning of the project, what the required development effort is, how much time that will take, and as a function of those factors, how much will the project cost. In short, the Chaos report measures the success of the project forecast to a much greater extent than it does the success of the project.

Eveleens and Verhoef (2009) found that organizational politics played a substantial role in their forecasting procedures, and that project management metrics are easy to manipulate. They found that an organization that had adopted the Standish definitions of project success produced significantly better results than the Standish report itself, but upon further review they found that this organization's forecasts were based upon the worst-case scenarios. Further, another organization that they reviewed scored very poorly based upon the Standish definition of success, yet was found to be the most accurate with initial forecasts.

Table 2.6. Standish CHAOS results

Year	Success	Challenged	Failure
1994	16%	53%	31%
1996	27%	33%	40%
1998	26%	46%	28%
2000	28%	49%	23%
2004	29%	53%	18%

Even so, project management forecasting quality remains important, as it plays a significant part in the decision as to which projects are to be undertaken, even in agile environments. However, because agile claims of impacts include assertions about higher quality, higher satisfaction, and additional organizational benefits, this study is also concerned with additional, alternative measures of project success.

Definitions of IS Success

Delone & McLean (1992) defined a model of IS success with the dimensions of system quality, information quality, use, user satisfaction, individual impact, and

organizational impact. This Delone & McLean model has been tested numerous times (Delone and McLean 2003; Petter et al. 2008; Rai et al. 2002), and has been found to have good explanatory power (Rai et al. 2002). Seddon (1997) proposed a modified model which removes IS Use from the causal nomological network, and Rai et al. (2002) showed that use occurred independent of user satisfaction, due to the fact that IS systems often become the only source for specific information, and access to that information requires use. However, Rai et al (2002) found that both the Delone & McLean and Seddon models showed significant explanatory power.

Both Rai et al. (2002) and Petter et al. (2008) state that the evidence regarding IS Success models validates the viewpoint that an integrated multi-construct measure of IS success is appropriate. Further, this measurement of IS success should measure beliefs, attitudes and behaviors and their interdependencies.

Thus project success in this study is conceived as a combination of project management success and the success of the IS project output in terms of system quality, customer satisfaction, and perceived benefits.

Contingencies In the Project Environment

Contingency theory was originally developed as a way to explain the relationship between organizational structure and performance. Contingency theory proposes that there is a fit between organizational structure and various external contingencies, and that the level of fit between the two is a driver of performance (Donaldson 2001; Lawrence and Lorsch 1967). Two core environmental contingencies that have been hypothesized as requiring adjustments for fit are environmental uncertainty and technology.

Burns and Stalker (1994) contrasted mechanistic structures, which were typified by hierarchical structures, and where task related knowledge and decision-making is centralized. In contrast, an “organic” structure is characterized by a network of empowered employees, who build a shared understanding of the task, and accept decentralized responsibility for delivery. They argue that high rates of change in technology or market environments require the use of organic structure.

Early versions of contingency theory focused on the impacts of contingencies on organizational structure and corporate strategy (Venkatraman 1989), but the theory has been applied to the contingent effects of new product design and software development, in which the impact of contingent factors on the performance of product development process performance is well documented (Barki et al. 2001). The theoretical model of this dissertation adopts the view that the development process chosen by a team may fit more or less well with the environment. While agile methodologies may be used in any development situation, it is proposed that because of its stress on multiple levels of feedback, and the repeated acquisition of feedback, agile method use impacts may be seen most in environments that are more uncertain. Further, uncertainty as provides key contingent factors that moderate the impacts of agile method use on project success.

The Components of Uncertainty as Contingent Factors

Projects are undertaken to deliver a unique product or service (PMBOK 2000). The practice of project management establishes specific processes to plan, execute, monitor and control projects, based upon numerous contingent factors. Major contingent factors in projects include the proposed size or scope of the project, the availability of

resources, financial, human, and material, and the time available to complete the project.

Based upon these factors, projects are more or less complex to manage. The appropriateness of any software method will be judged on its ability to complete the project to which it has been applied. As described above, most projects are challenged or fail, according to traditional project management metrics. As such, traditional software methodologies may be considered a poor fit for the projects to which they have been applied. However, several particular project & product characteristics have been proposed as being specifically important when choosing software methodologies.

“Home grounds” have been proposed for agile and plan-based approaches (Boehm and Turner 2004). Boehm & Turner contrast the primary goals of agile vs. plan based approaches, and state that there are specific project, product and environmental characteristics that indicate more appropriate fit for the use of one type of method vs. the other. Their five factors are project size, project criticality, environmental dynamism, personnel, and culture and are listed here as Table 2.7. This table provides a list of the kinds of contingencies that may affect the fit between the environment and development method.

Size: As a project becomes larger, due to greater scope, team size, or both, communication becomes more difficult. Methodologies that rely on building a shared mental model of a problem and its associated solution will face obstacles as the size of the problem to be modeled becomes larger.

Criticality: Boehm and Turner (2004) claim that as a system’s criticality rises, plan-driven approaches are a better fit. This is likely caused by the growing objective

nature of requirements on life-critical systems vs. process-automation systems of lower criticality (see Table 2.7).

Dynamism: In highly dynamic environments, up front planning does not provide opportunities to learn from feedback, to make sense of the environment, and to generate high hit potential (MacCormack and Verganti 2003).

Table 2.7. Five Factors Determining Fit of Agile vs. Plan-Based Methodologies

Factor	Agility Discriminators	Plan-Driven Discriminators
Size	Well-matched to small products and teams. Reliance on tacit knowledge limit scalability	Methodologies evolved to handle large products and teams. Hard to tailor down to small projects.
Criticality	Untested on safety-critical products. Potential difficulties with simple design and lack of documentation.	Methodologies evolved to handle highly critical products. Hard to tailor down to low-criticality products.
Dynamism	Simple design and continuous refactoring are excellent for highly dynamic environments, but a source of potentially expensive rework for highly stable environments.	Detailed plans and Big Design up Front excellent for highly stable environment, but a source of expensive rework for highly dynamic environments.
Personnel	Requires continuous presence of a critical mass of expert resources.	Needs a critical mass of scarce expert resources during project definition, but can work with fewer later in the project – unless the environment is highly dynamic.
Culture	Thrives in a culture where people feel comfortable and empowered by having many degrees of freedom (Thriving on chaos)	Thrives in a culture where people feel comfortable and empowered by having their roles clearly defined by clear policies and procedures. (Thriving on order)

Personnel: Alistair Cockburn's (2001) "levels" of developer skills define three skill categories of developer. Level 1 developers can only, with training and experience, act in well defined situations. Level 2 & Level 3 developers have the understanding of methodologies, practices, and corresponding rules to either modify or break the rules to

align the method with emergent, unexpected and unprecedented situations. As agile methodologies are intended to be flexible and reactive to unexpected situations, Boehm & Turner propose that development methodologies must retain a higher level of resource throughout a longer portion of the project than plan-based methodologies.

Culture: Finally, the adoption of a cultural philosophy that values the flexibility and potential value of allowing change is key to successful adaptive product development (McAvoy and Butler 2009).

The Research Literature on the Impacts of Agile development

The research literature on the impacts of agile development is still relatively new. Even as web searches find literally hundreds of presentations and papers presented at conferences, as of 2008, only 35 empirical studies of rigor had been published, and of those, only 6 had appeared in journals (Dyba and Dingsoyr 2008). Further, a number of those studies that had been completed were descriptive, focused on why organizations might adopt agile development, or compared agile to traditional development processes. Since then, several special issues have focused on agile development. However, even as research on agile development has continued to grow, the majority of the published work does not address empirical observations of agile development impacts.

Table 2.8 lists a selection of recent scholarly literature on the impacts of agile development. There has been a great deal of literature that has focused on agile development practices, a number of pieces that have applied the lens of agile to certain aspects of organizations, and a few articles that have attempted to tie agile development into the previous literature on control and organizational learning.

However, to my knowledge, relatively few examples exist of research that attempts to create a holistic theory of agile development adoption and impacts.

Maruping et al. (Maruping et al. 2009a; Maruping et al. 2009b) produced two recent articles that integrated agile method with the control and expertise coordination literature. They measured XP practices including pair programming, collective code ownership, refactoring, and continuous integration, and hypothesized that these practices would increase software quality, as indicated by the number of software bugs identified. They identified two contingencies that inform the theoretical model adopted .

First, they treated requirements change as a moderator, in the presence of which the impacts of agile method use increase. Second, they investigate two control phenomena, outcome control and self control. They found that there is a significant interaction in how these factors affect agile development impacts. Outcome control increases the impact of agile methodologies when requirements volatility is high, but not when it is low. This is intuitive, based upon the discussion of team adaptability above. Outcome control is more appropriate in highly volatile environments since outcome controls give teams a general goal toward which to work, while providing the team with the authority and autonomy to determine the best way to achieve the goal. When a process output is not novel, or can be well defined in advance, the steps necessary to complete the process are likely to be definable, leading to the likelihood of the applicability of behavioral control.

Maruping et al. (2009a) found that the construct of output control moderated the impact of agile method use on a set of software quality measures. However, their measurement of output control involved relatively vague outcome indicators such as

“Significant weight will be placed upon timely completion”, and “Project goals were outlined at the beginning of the project”. As output control specifies the need for well-understood and clear indicators of task outputs, it is unclear if these indicators measured outcome control.

Maruping et al. (2009b) measure the use of 2 XP practices, collective code ownership and coding standards on software quality, as measured by number of bugs. They find that these relationships are significant, and additionally that the impacts of expertise coordination are lowered on teams that use these practices.

These two studies are constrained by their narrow conceptualization of project success, and their treatment of agile method use as indicated solely by the level of adoption of particular XP practices. Additionally, the teams being measured had implemented a wider set of practices than were included in the model in Maruping 2009b. Further the practices are proposed to have heavy interaction effects when used together. Therefore the impact of the particular processes on the dependent variable is difficult to extract from the impact of the use of XP in general.

Harris et al. (2009) used case study research to extend the control literature with a modification to the concept of outcome control. As described previously, they recognized that outcome control occurs at the end of a project. Since it is based upon criteria developed at the beginning of the project, it is not aligned with agile development philosophy. They illustrate that turbulent environments require the application of flexible capabilities, and that a priori understanding of both the goal of a project, and the evaluation of project results is made more difficult based on uncertainty.

As they explained, output control is not effective in an iterative and emergent development context because it is seen as an evaluation measure of an entire process or project, is performed one time, at the completion of task or process performance, and the evaluation is performed based upon an a priori definition of process requirements. Table 2.7 compares output and emergent outcome control.

Table 2.8. Recent Empirical Literature on Agile Methodologies

Study	Description & Research Method	Key Constructs	Agile Method Tested (if specified)	Agile Practices Measured	Contributions
Salo and Abrahamsson (Salo and Abrahamsson 2008)	Descriptive survey of European XP and Scrum. Survey, 35 projects over 18 organizations	N/A	XP and Scrum	XP (12) Scrum (5)	Found that agile methodologies are generally perceived by the development team as helpful.
Mann (Mann and Maurer 2005)	Studied the impacts of deploying Scrum in a single organization. Case study – single organization over two years.	N/A	Scrum		
Maruping, et al. (2009a)	Studied the impact of agile practices on teams' ability to respond to requirements change. Longitudinal Survey of 862 members of software development teams within a large U.S. Consulting organization	Agile Method Use hypothesized to positively impact Software Project Quality Moderators: Requirements Change Outcome Control Self Control	XP	Pair Programming Refactoring Coding Standards Collective Code Ownership Continuous Integration	Found that agile method use is most important in improving project quality when outcome control and requirements change are high.

Table 2.8. (cont'd)

Study	Description & Research Method	Key Constructs	Agile Method Tested (if specified)	Agile Practices Measured	Contributions
Harris, et al. (2009)	Studied the impact of XP in turbulent environments. Introduced the concept of emergent outcome control. Three case studies.	Emergent Outcome Control	XP	None	The concept of emergent outcome control as an extension of control theory.
Austin & Devin (2009)	Theory of agile as post-industrial production. Inductive Argument	Post-Industrial Production is theorized to be possible due to the fact that technology now exists that allows the cost of novelty to be reduced to a level below the benefits of novelty	N/A	None	Made a theoretical argument that enabling technologies might allow for novel development to be inexpensive enough to be feasible.

Table 2.8. (cont'd)

Study	Description & Research Method	Key Constructs	Agile Method Tested (if specified)	Agile Practices Measured	Contributions
Lee & Xia (2010)	Conceptualized "agile" as team response extensiveness and efficiency. Focused on the idea that software team autonomy & diversity led to greater response to change. Survey of 565 project managers	On-Time Completion On-Budget Completion Software Functionality	N/A Did not measure any particular agile practices or culture.	None	Re-conceptualized Agile as the level of autonomy enabling response to changes. Team diversity is not mentioned by agile alliance. Team autonomy may be interpreted as Principle 5
Fruhling & De Vreede (Fruhling and Vreede 2006)	Studied XP in an environment building an emergency response application. Used action research to view the 12 XP principles in practice, with the intention of operationalizing the practices.	Utilized the four agile values as a lens to describe the level of agile adoption.	XP	All XP practices	Descriptive study - Illustrated the use of XP in practice, how they could be operationalized.

Table 2.8. (cont'd)

Study	Description & Research Method	Key Constructs	Agile Method Tested (if specified)	Agile Practices Measured	Contributions
Lytinnen & Rose, 2006	Viewed agility via an organizational learning, exportation/exploitation lens. Longitudinal Case Study	Type 0 innovation (by others) Type 1 innovation (new ways of building IS) Type 2 innovations(innovative new IS) Outcome Variables: Innovative Content, Speed Cost	N/A	None	Gave agility a more multi-faceted definition. Described the importance of adoption of innovative technologies as an enabler of agile innovation.
Maruping, et al. (Maruping et al. 2009b)	Studied the impacts of two XP practices (collective code ownership and coding standards) on software technical quality. Survey of 56 project teams consisting of 509 software developers.	Expertise Coordination Collective Code Ownership Coding Standards Software Technical Quality (lack of bugs)	XP	Collective Code Ownership Coding Standards	Found that collective code ownership and coding standards moderated the impact of expertise coordination.

Table 2.8. (cont'd)

Study	Description & Research Method	Key Constructs	Agile Method Tested (if specified)	Agile Practices Measured	Contributions
Cao, et al. (Cao et al. 2009)	Looked at the adoption of agile through the lens of adaptive structuration. Found that structure emerged that both enabled and constrained the adoption of agile practices. Case Study.	Challenges to the adoption of agile Customer Challenges Developer Challenges Management Challenges	XP	All XP practices	Illustrated the impacts of social structure on success of agile projects.
Mangalaraj, et al. (Mangalaraj et al. 2009)	Using the lens of software process innovation diffusion, looked at factors that impacted the adoption of agile practices. Case Study.	Acceptance of XP Practice	XP	All XP practices	Found individual, team, technological, task, and environmental factors to influence the acceptance of XP practices in an organization.
Port, et al (Port and Bui 2009)	Using simulation, the authors test the ability of plan-based methodologies and agile methodologies to respond to requirements change.	Requirements volatility Requirements prioritization process (agile or plan-based)	None (General concept of prioritizati on)	Not specified	They found that, consistent with Boehm & Turner (2004), that requiremetns volatility was a key driver of success. When volatility was high, agile methodologies performed better than plan based, and vice-versa.

The key differences between traditional output control and emergent outcome control are shown in Table 2.9. Emergent outcome control is executed repeatedly throughout the project, is used as a method for correcting the trajectory of the project deliverables, and the "definition of done" (in scrum terminology) evolves and emerges during the project.

Austin & Devin (2009) describe the cost of novelty in "post-industrial" making. They contrast the making processes of pre-industrial vs. industrial making, and illustrate how the pre-industrial (craft) making process allowed for the delivery of high levels of uniqueness, or novelty, but at a high cost. Contrasting with this, industrial making is best at producing low-cost products of low variance. They make the assertion that novel outputs will only be sought when the cost of producing them is lower than the benefits.

Table 2.9. Output control vs. emergent outcome control (Harris, 2009)

	Purpose	Frequency	Evaluator	Construction of Standard	Comparison
Output	Evaluation	Once	Supervisor	A priori	Completed project versus specification
Emergent Outcome	Corrective Action	Continuous	Multiple stakeholders	Evolving by stakeholder	Emergent coutcomes versus tacit specifications

However, they make the claim that technology may enable the emergence of post-industrial making, just as it enabled the production processes of the industrial revolution. Because digital technologies can reduce the reconfiguration and exploration costs of software development, they propose that enabling technologies may drive the cost curve of novelty lower, allowing for the more frequent delivery of novel outputs.

Although they make this claim, they do not develop it, nor do they provide illustrations of which technologies might lower this cost curve. However, as described above, agile development practitioners make several claims about the ways that technologies enable refactoring and other processes that are central to agile development.

Cao et al. (2009) and Mangalaraj et al. (2009) both investigate impacts of the environment on agile development adoption and impacts. Cao et al. adopt the lens of adaptive structuration theory to describe the effects of XP implementations, and the social structuration that took place. They found that there were significant structural barriers to the use of agile methodologies like XP. For example, they found that upper management was unwilling to proceed with development without a strong up front estimate of the project cost. This led to the team not adopting the planning game practice, and instead adopting a plan-based estimation process. At the same time, management did agree that based upon the experience during a pilot XP project, that cost estimation was likely to change during the project, and that cost estimate changes would not be used as an indication of project failure.

Mangalaraj et al. (2009) used the lens of innovation diffusion to investigate the acceptance and adoption of various XP practices. Within a single organization that was adopting XP, they observed two projects, one a new development effort, and one an ongoing development team for an existing system. They observed that the two teams (X & Y) had significantly different reactions to the decision to adopt XP. Team X exhibited significant adoption of XP, while team Y resisted the implementation. They identified five categories of barriers to and enablers of adoption of new development practices, individual, team, technology, task, and environmental. While this study investigated

agile development innovation adoptions, the contingent impact of factors such as these on project outcomes can be interpolated.

At the individual level, they found that team X's Knowledge of XP processes gave Team X an understanding of the purpose and interplay of the XP processes. Team Y, even though they were trained on XP in the same fashion as team X did not retain knowledge of XP practices, how to perform them, or what purpose they served. Team X had a very positive attitude toward XP, while team Y had a very negative attitude. At the team level, Team X adopted an XP egalitarian and empowered perspective, while Team Y maintained their previous, hierarchical roles.

Consistent with Austin & Devin's (2009) proposition, and the practitioner literature on agile development, Mangalaraj et al. (2009) found a significant impact of technology factors on both agile development adoption and impacts. These fell into two main categories, compatibility, and tools support.

Compatibility - Team X utilized Java as the sole language for development. This single technology made utilizing collective code ownership possible, and contributed to velocity, while Team Y was constrained by the legacy technology that hindered their ability to collectively manage the code. Rather, the team members were more or less expert in their use of the various legacy coding platforms and tools, and ownership & specialization developed because of this. Additionally, the ability to pair program was constrained by the same issue.

Tools Support – Team X utilized tools that supported continuous integration, refactoring and test-first programming, while Team Y could not, due to their legacy technology. "These tools greatly helped members of project X in adhering to XP

practices. This notion was echoed by several members of project X. According to one member, '[tool] is a big factor in being successful. It greatly helps in refactoring the code quickly and safely.' Another member said, 'Tool support is definitely important and it varies according to the language. Without [tool]-like tool it may be hard to do refactoring.' In contrast, the tools and Integrated Development Environments employed in project Y offered little or no support for core XP practices. Because of this, members of project Y faced great difficulty in implementing some of the practices and had to impro- vise. According to a team member, 'Test first in C++ platform is difficult to implement; for Java they have [tool] and here we tried it but we could not'" (Mangalaraj et al., 2009 p 350).

Task level factors also emerged in this study. Team X built a green-field system, consisting of 30,000 lines of code in less than 16 months. Team Y on the other hand had two significant constraining task factors. First, they had a large legacy code base to support. This code base was built on three disparate technologies, had no automated regression test code, and had been developed over ten years. While Team X could make refactoring moves at will, Team Y could not, for fear of breaking the system. Additionally, while Team X's system required little integration with external systems, Team Y's system required extensive integration, limiting its ability to act in an agile fashion.

Finally, the authors identified three environmental factors, budget constraints, time constraints and customer participation. The team's customers and management had significantly different methodologies of control. While team X's management seems to have used primarily output control, Team Y's external control was performed by

“demanding” product managers, who “dictated” their time and budget constraints. Because of this, pairing and refactoring (the modification of code to make it of higher quality, while leaving functionality unchanged) were viewed as wasteful.

Lee & Xia’s (2010) study views the impacts of team agility as indicated by team response efficiency and extensiveness (Lee and Xia 2010). They conceptualized the use of agile method as the level of autonomy granted the team. This conceptualization was used as an independent variable to test its impact on team’s response to changing requirements. They found that team autonomy led to the ability to respond effectively to changing requirements, which increased project performance.

Finally, Port & Bui (2009) used simulation to test the agile vs. plan-based processes for requirements prioritization. They found that agile development requirements prioritization processes outperform plan-based processes in situations of high requirements volatility.

Agile Methodologies: Structure and Impacts

Based upon the discussion of the principles of the agile manifesto, the importance of feedback, the discrepancies between agile methodologies, and the previous research described above, it is clear that agile methodologies are a complex phenomenon. While much of the previous literature has focused method-specific practices, it is encouraging to see that more broad based constructs are emerging. However, it is believed that, based upon the previous literature and prior theory, a case can be made for opening the black box of agile to a degree, and theorizing regarding the relationships between agile constructs and the nature of their impacts.

Figure 2.5 illustrates a theoretical perspective that integrates the prior literature into a holistic model of agile method use and impacts. As can be seen, the constructs that are likely to directly impact project success are separated from those likely to enable the direct effects.

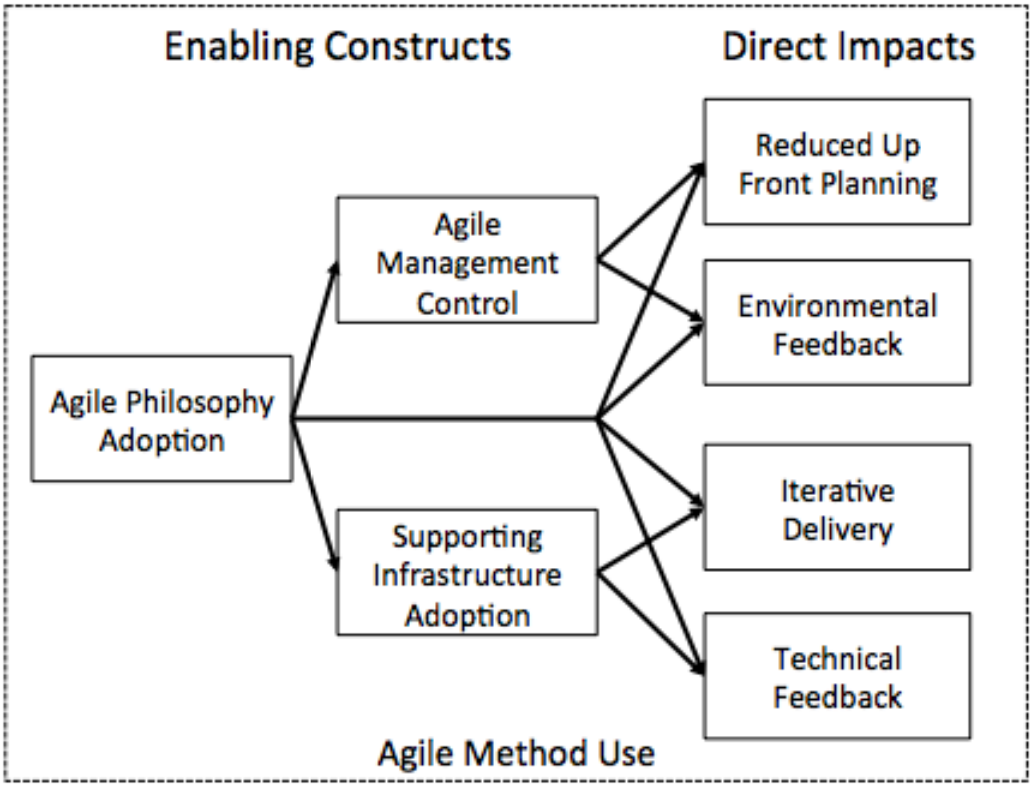

Figure 2.5. The Nomological Network of Agile Method Adoption and Use

Agile Philosophy Adoption

Because all methodologies are motivated by an underlying philosophy, adoption of an agile philosophy by the delivery organization will motivate the use of agile processes and practices, and the adoption of supporting technologies. However, philosophy adoption is not theorized to directly impact project performance. Rather, the philosophy motivates a congruent set of practices and processes that are hypothesized by the organization to reinforce each other toward a desired goal. In agile development,

this goal is to develop a dynamic capability to sense and respond to salient environmental cues, and to learn over time. This desire to quickly respond while learning requires both the support of management, and suitable technology support.

Agile Management Control

As discussed above, traditional control modes, particularly managerial control and outcome control, are incongruent with an agile method approach. Managerial control stresses the empowerment of management over the team to direct the team on actions to take, and to reward them for compliance. Outcome control is based upon the concept that the result of the project is well-definable by the team at the outset, and that proper incentives and rewards can be defined based upon an up front plan. Agile methodologies on the other had, due to their philosophical belief in an empowered team and in the empirical nature of the development cycle reject these modes of control. Instead, agile teams have adopted new control modes that are congruent with the concepts of agile. methodologies Emergent outcome control, based upon “mid-course” correction, supports the team’s goal of adaptive goal orienting throughout the project.

Supporting Infrastructure Adoption

Austin & Devin proposed that the agile methodologies indicate a break from industrial making, into an era of post-industrial making. They illustrated how the emergence of certain technologies enabled the industrial revolution and the principles (philosophy) of industrial making. They also proposed that certain technologies most likely enable post-industrial makers to make the cost of novelty lower. Further, they conjectured that these supporting technologies might shift the cost curve of the making of unique deliverables (Austin and Devin 2009). The agile movement has been enabled

by the serendipitous emergence of multiple, complementary supporting technologies, and that these technologies, described below, are a key component of agile teams' agility. Several technologies that directly impact a team's ability to adaptively sense and respond are Source Code Control, automated testing software, and continuous integration software. We propose that the three constructs described above are enabling processes for the feedback processes that directly impact the team's ability to deliver projects successfully.

Figure 2.8 presents the full theoretical model adopted for this study. Although the full theoretical model of agile methodologies and their structure was presented here, the research study that was undertaken in this dissertation focuses solely on testing the four feedback-related constructs for which direct impacts are proposed. Those constructs are described and elaborated in Chapter Three.

This model proposes a richer conceptualization of agile method adoption than has appeared previously in the literature. This conceptualization proposes that agile method adoption consists of more than the use of practices defined by agile practitioner. Three additional components, the adoption of an agile, innovative philosophy which informs and drives the use of the practices, the use of enabling technologies, which reduce the cost of operating in an iterative, adaptive manner, and congruent, agile management controls are key components of agile adoption.

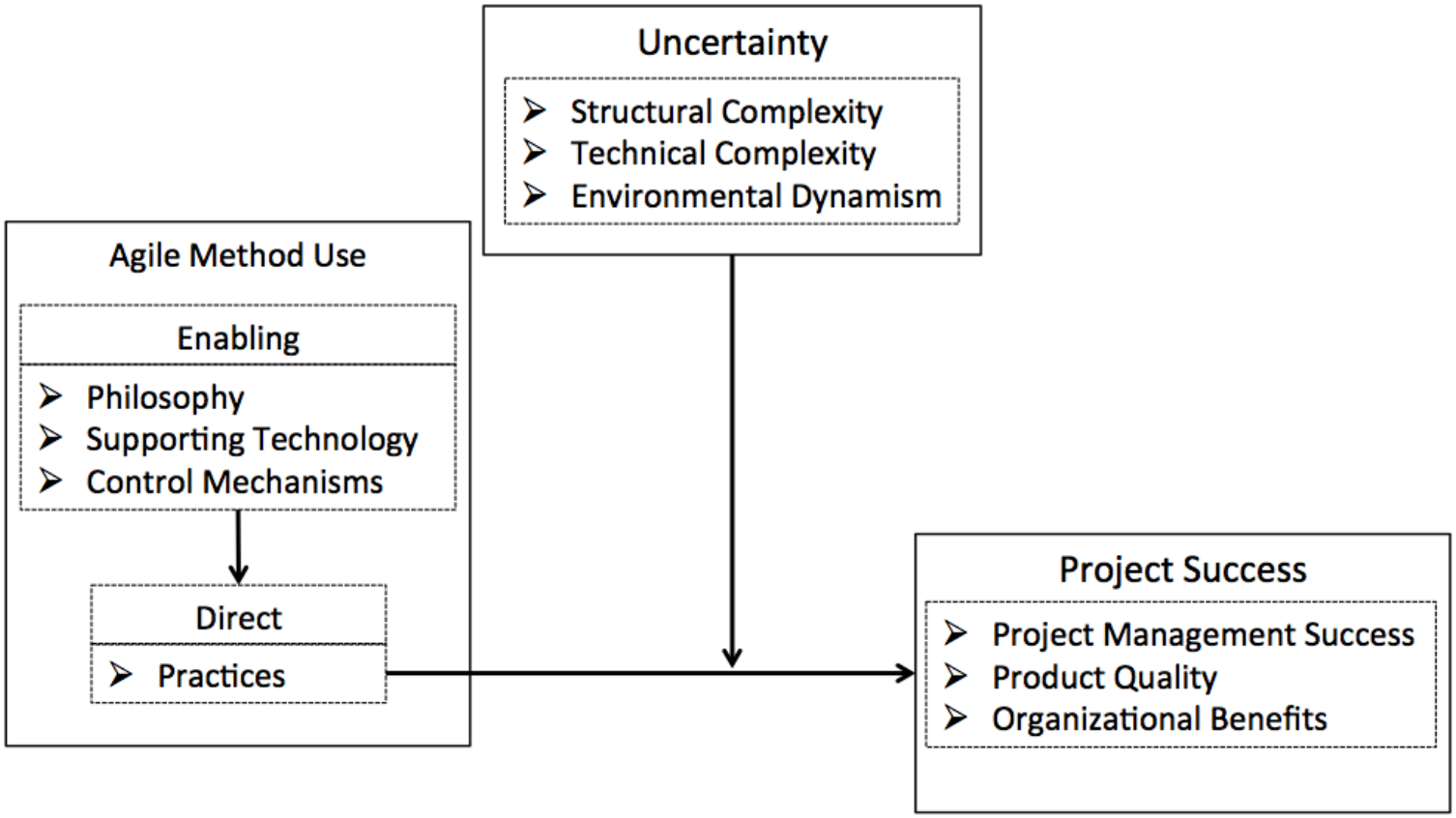

Figure 2.6. The Theoretical Model of The Impacts of Agile Method Adoption and Use on Project Success.

Chapter Summary

This chapter presented relevant literature from the organization theory and information technology literatures to support a theoretical model that is useful for conducting research on the impacts of agile method adoption on project success. This chapter illustrated that treating agile methodologies as a "black box" is inappropriate. We have presented a nomological network that describes the structure of agile method adoption and use as consisting of a set of reinforcing constructs. These constructs fall into two categories, one that impact project outcomes directly, and one that supports or enables these direct impacts.

Chapter three describes the theoretical model of the direct impacts of agile methodologies, which will drive the research study performed in this dissertation.

Chapter Three: Research Model

Introduction

Chapter two developed a theoretical structure that described a framework for understanding the adoption and use of agile methodologies as being a two-level structure, with the adoption of agile philosophy, agile management control, and agile supporting technology supporting the ability to perform practices that would directly impact project success. Further, a conceptualization of agile practices was described that is based upon the feedback mechanisms built into agile methodologies, rather than the specific engineering and process practices of particular agile methodologies. The full testing of this theoretical model is beyond the scope of this dissertation. Instead, this study focuses on the ability to test the impacts of agile practices posited as being most likely to directly impact project outcomes. In this chapter, a research model for testing the direct impacts of agile practices on project success is presented. The empirical portion of this dissertation addresses the following research question:

How do agile methodologies impact project success?

Research Model and Variables

Prior IS research has identified that the nature of development method impacts are contingent. This means that when there is higher fit between the method itself and the class of problem being solved, higher performance is predicted (Avison and Taylor 1997; Barki et al. 2001). As uncertainty rises, method flexibility becomes a required component of fit, and of potential positive project outcomes (Barki et al. 2001; MacCormack and Verganti 2003; Miller 1992). Further, agile methodologies are claimed to be most appropriate when used under uncertainty.

The research model proposed in this chapter is essentially an operationalization of components of the theoretical framework exposited in Chapter Two. Figure 3.1 shows the research model.

The unit of analysis of this dissertation is the individual software project. The key measure of success is the delivery of a particular project deliverable. A summary of the variables presented in this model is provided in Table 3.1.

In the following sections, the constructs of the research model are described, including the conceptualization of agile method and use and project success, and utilize the theoretical lenses previously discussed to motivate hypotheses of the impact of agile method use on project success.

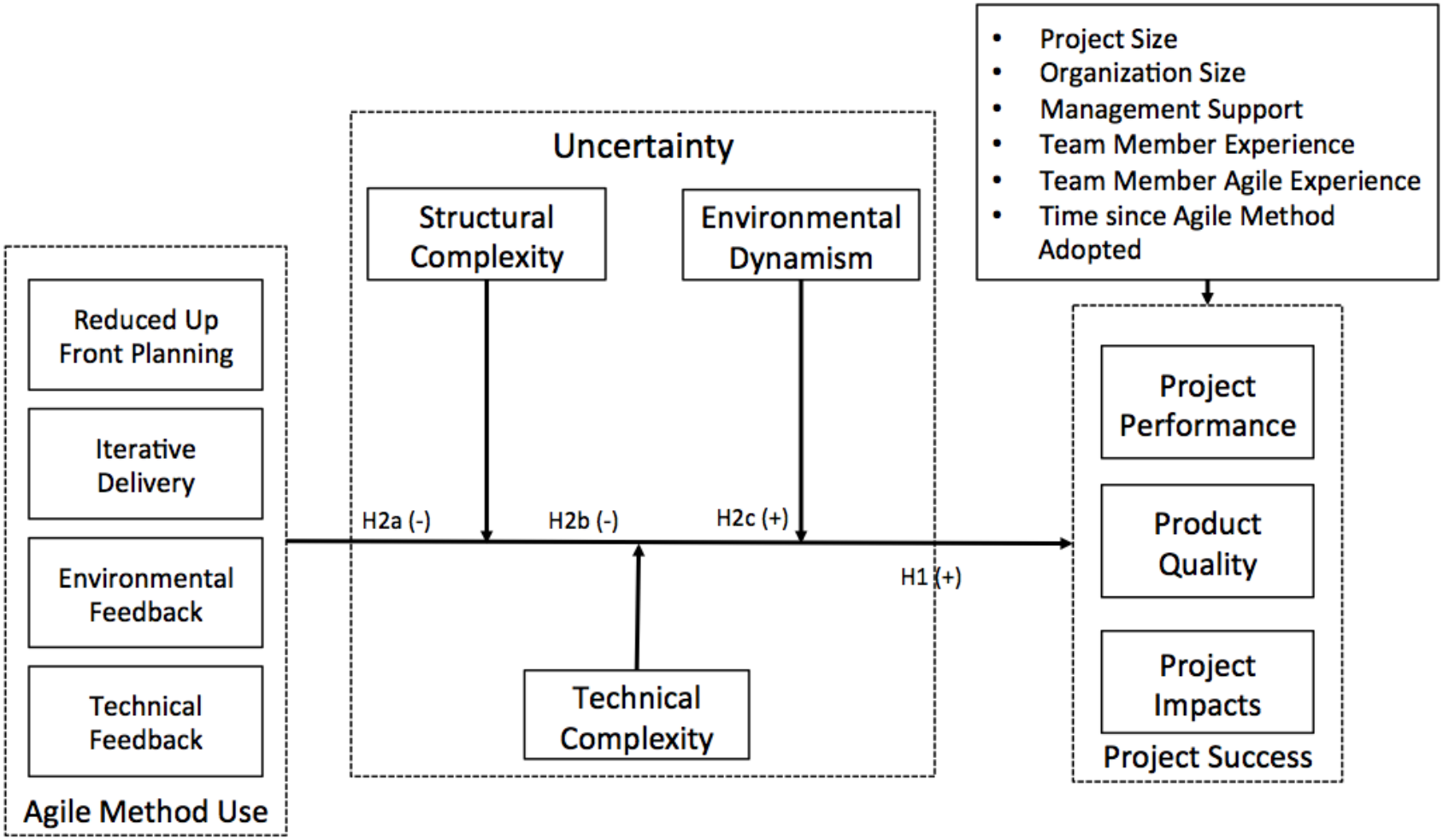

Figure 3.1: The Theoretical Model for Analyzing The Impacts of Agile Method Adoption on Project Success

Table 3.1. Research Variable Definitions

Dependent Variables	
Project Management Success	The level to which project results match defined goals of scope, schedule & budget.
Product Quality	The perceived quality of the system as defined by perceived quality, effectiveness, completeness, reliability, suitability, and accuracy.
Project Impacts	The level to which the project is perceived to have positively impacted the organization, as indicated by perceived benefits and perceived satisfaction.
Extent of Use of Agile Methodologies	
Reduced Up Front Planning	The level to which the team reduces the time spent before beginning work.
Iterative Delivery	The level to which the team delivers functional code each iteration.
Environmental Feedback	The level to which the team utilizes mechanisms to obtain feedback from customers and stakeholders.
Technical Feedback	The level to which the team utilizes mechanisms to ensure that the system is functioning properly.
Moderating Variables - Uncertainty	
Structural Complexity	Index of size of team, criticality of project, number of stakeholders, and team distribution (organizational and geographical)
Technical Complexity	The level of complexity of the system, and complexity of integration with other systems.
Environmental Dynamism	The level of requirements, technical, and organizational change during the project.

Table 3.1. (cont'd)

Control Variables	
Project Size	The size of the agile development project, as indicated by the number of people on the agile development team.
Organization Size	The size of the organization served by the agile team.
Management Support	Willingness of top management to modify practices and provide support to enable agile adoption.
Team Member Experience	Number of years development experience.
Team Member Agile Experience	Number of years experience in agile teams.
Time since Agile development adopted	Number of months since organization adopted Agile development.

Project Success

Delivering the promised scope of a project, within time and budget constraints, is the core of project management. Agile methodologies claim to better manage the impacts of uncertain environments, and to deliver projects more successfully. The negative impact of uncertainty on project management success is well documented (e.g., Keil et al. 1998; Nidumolu 1995; Schmidt et al. 2001; Wallace and Keil 2004; Xia and Lee 2005). IS project failure is particularly common, due to the delivery of programs that do not deliver useful features, are delivered late or not at all, or escalate out of control (Keil 1995; Lyytinen and Hirschheim 1988).

Project success has been viewed as a multi-dimensional construct, including the performance of both the software development process and the performance of the product itself. Project performance means the level of performance against (1) project management metrics, (2) product quality, and (3) perceived impacts of the product. It is appropriate that they are measured as separate constructs, because they are not

expected to correlate highly. For example, while a project might be delivered significantly over budget, that particular project may at the same time deliver a product of very high quality.

Most projects, whether utilizing agile methodologies or not, are justified based upon some set of cost and duration and feature set (scope) estimates established before the project begins. These project estimates assume the ability to deliver a suitable system within the constraints of this business-driven timeline. Even though agile methodologies attempt to minimize up-front planning, agile methodologies have been reported to deliver projects within high level estimates generated at the beginning of projects (Coad et al. 1999; Highsmith 2002). Because agile methodologies make claims of high rates of delivery within established project constraints, one dimension of project success in this model is project management metric performance.

Agile methodologies also make claims that they delivery more useful software that better reflects the needs of the customer. In short, they claim that they develop high-quality software. Because agile development projects involve quick iterative feedback, the progress of a project is communicated much more transparently than within traditional development environments. Agile development teams strive to produce working software as their progress metric. Because agile development teams develop priorities with their users within the short iteration cycle, adaptation occurs rapidly and repeatedly throughout the project. This adaptation enables the agile team to identify required changes to plans, develop an emergent understanding of the real needs of the customer, and sense and respond to incongruity between the emerging system and the required system.

Finally, although it is not expected that formal organizational impacts such as ROI will be identifiable for most projects, a successful project will most likely be accompanied with perceived benefits to the user and organization. Because, agile development teams value frequent, direct interaction with users in order to develop a shared understanding of the business problem and the associated solution requirements. Because of this, the product of an agile development project should exhibit a stronger congruence with organizational requirements, and should be perceived as delivering a significant, positive impact to the organization.

Extent of Agile Method Use

As previously noted, very few of the particular practices advocated by agile practitioners are truly novel, and most agile development practices are suitable to be applied outside of an agile project setting. Because of this, the fact that a team performs agile practices does not necessarily indicate that it has adopted an agile method (Beck 1999; Highsmith 2002). Therefore, to measure the extent of agile method use, the generalized set of constructs defined in Chapter Two was used: Reduced Up-Front Planning, Iterative Delivery, Environmental Feedback, and Technical Feedback.

Project feedback events occur naturally and intentionally. As described in Chapter Two, four generalized processes are most likely to provide feedback to the team, and drive project impacts.

Reduced Up Front Planning

Reduced up front planning is likely to directly impact project performance in two ways, by reducing the time to initial feedback, and by reducing waste from planning tasks too far in advance, leading to rework. The manifesto argues that it is more

important to deliver software than the deliverables that are generally created at the beginning of projects. The agile methodologies surveyed all placed a premium on reducing the amount of time spent on up front planning. Speed of initial feedback has been shown to increase performance under uncertainty (MacCormack 2001), and reducing the window of feedback triggers natural feedback assessments to occur more quickly (Gersick 1988).

Further, agile practitioners argue that the software development process is highly uncertain, and the process of completing documentation of system requirements up front is likely to be wasted effort for tasks scheduled for completion far in the future.

Environmental Feedback

Agile methodologies are founded on the premise that environmental feedback is key to a project's success. The agile methodologies prescribe numerous practices meant to generate environmental feedback, including the retrospective, the daily stand up, the on-site customer, and the informative workspace among others. These practices reflect the agile manifesto's call for direct, face-to-face interaction and their philosophy that recognizing the need for change, and facilitating change is more productive than attempting to restrict change (Fowler and Highsmith 2001).

In uncertain environments, feedback is the source data that allows teams to sense and respond to the environment (Burke et al. 2006). If, as described above, the software development process is inherently uncertain, access to and processing of environmental feedback should positively impact project performance (Eisenhardt and Tabrizi 1995).

Iterative Delivery

The first principle of the agile manifesto is “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.” This entails an unstated corollary, which is that the software is deployable at an early stage, and continues to be deployable throughout the delivery schedule. When software is delivered iteratively, it can be deployed and utilized much earlier in the project.

Iterations consist of short, time-boxed deadlines, usually no longer than 4-6 weeks (Highsmith 2002; Schwaber 1996; Schwaber and Beedle 2002), and often much shorter. While prototyping and other evolutionary methodologies emerged before agile methodologies, agile methodologies are differentiated in that, in most cases, the agile iteration structure include all or most of the steps of the waterfall method. For the requirements selected for the iteration, the solution is designed, built, tested, and *delivered* (Beck and Andres 2004; Highsmith 2002; Schwaber and Beedle 2002). While this does not necessarily mean that the code is deployed to production, the understanding is explicit in several methodologies that the code should be deployable at the end of any iteration, at the discretion of the user (Beck 1999; Highsmith 2002; Schwaber and Beedle 2002).

By using this this delivery model, agile teams reduce the timeframe of definitive feedback, which allows them to respond more quickly to uncertainty that stems from environmental dynamism (Duncan 1972; Lawrence and Lorsch 1967). By working on small parts of the system in short timeframes, the potential impact of requirements changes is limited to those requirements being worked on in the current cycle. By delivering working software each cycle ambiguity in requirements is reduced, as users

provide feedback about a working system, not an abstract requirements document. These feedback cues are of high quality because, instead of being based upon representations of the system (as would occur during the analysis and design phase of a waterfall project), they are based upon a review of the working software. This working software is a far richer medium for users to evaluate whether the system that is being built is suitable for the task (Brooks 1987).

Technical Feedback

Technical feedback is achieved in agile methodologies by technology-mediated processes. The technology-mediated processes are source code control, automated testing and continuous integration.

Source Code Control, or Software Configuration Management (SCM) provides developers with the ability to manage the full repository of system components, via versioning, file control, and configuration (Estublier 2000). In addition, SCM systems provide build and rebuild support, reduce the need for developers to manage dependencies between files, and allows for teams to collaboratively and cooperatively build software via check in and rebase (download changes made by others) functionality. Most importantly, SCM systems allow for the identification of groups of changes made at the same time, and allow the rollbacks of single or groups of changes easily.

Automated Testing software enables software teams to develop libraries of test code and run them repeatedly. Developers might have previously executed tests on an ad hoc basis. Automated testing frameworks allow teams to write suites of test code that cover the unit, system, integration, and user interface levels of testing. These tests

are standardized via a framework, and can be run automatically with little effort or special training. Teams can enforce routines that ensure that all the tests, covering the entire system, must be run before code is checked into the repository. Previous to the emergence of automated testing platforms, teams often relied on familiarity with the software and technical environment to predict the impacts of a change. This familiarity helped developers to be more efficient in testing and implementing larger changes because they know what needs to be tested and how to implement the changes (Curtis et al. 1988).

Eisenhardt and Tabrizi's (1995) findings presented in Chapter Two suggest that the speed of testing directly impacts the speed of project completion due to the early feedback that the team gets as to product failure. As automated testing leads to near continuous testing of the entire product each day, it should be expected that project success will be enhanced.

Historically, one of the most time consuming and problematic activities of software development teams is the integration of various code modules into a single, working build. *Continuous Integration* (CI) is the process of integrating the entire code base in an automated fashion, as often as possible, and successfully (Fowler 2006). While automated testing can enable team routines that require testing before check-in, issues can arise if a single developer doesn't remember to re-base (refresh the code base locally) before testing, or if it is not feasible to perform local unit testing cannot be used to test integration to external test systems. In these cases, a centralized, automated build process can be used to ensure that the full code base is functional at all times. The presence of automated build software, enabled by the presence of

automated testing frameworks and software configuration management, enables agile development teams to implement practices, such as CI that otherwise could not be implemented. These practices directly impact the ability for the development team to respond to the feedback cues that the environment provides, and provide specific technical feedback to the team.

The previous discussion motivates the first hypothesis:

Hypothesis 1: The extent of agile method use will positively impact project success.

Uncertainty and Project Success

Uncertainty, as indicated by complexity and dynamism, moderates the impacts of agile methodology use on project success. We theorize that the moderation effects of uncertainty will present themselves differently by the component of uncertainty.

Turning first to complexity, it is theorized that increased levels of complexity will impede the ability for a team to both process and respond to salient environmental cues. As described above, two dimensions of complexity salient to software development are structural complexity and technical complexity.

Structural Complexity

Structural complexity is characterized in this study by (1) the size of the project team, (2) the number of reporting units within the team, (3) the geographic distribution of the team, and (4) the criticality of the project. As project organizations grow in both size, and in the number of reporting units represented, the number of stakeholders increase (Baccarini 1996). Further, structural complexity rises due to the geographical distribution of the team (Hinds and Mortensen 2005), and the criticality of the project (Xia and Lee

2005). Together, high levels of these structural elements increase both the complexity of requirements definition and coordination, and the cost of communication between team members. The presence of multiple stakeholders reduces the ability for the project team to clearly identify the most critical requirements, and higher coordination costs impede the team's ability to bring the full capabilities of the team to bear on the problem.

As the number of stakeholders and the cost of communication rise, agile teams are likely to be particularly negatively impacted. Agile development teams respond to continual feedback by making repeated updates to the shared mental model of the team. This shared mental model contains the understanding of the current environmental situation, current priorities, agreed upon standards of behavior, and codified team response plans to specific issues that arise.

The development of a shared mental model requires the team to process the meaning of cues together. Structural complexity in the form of multiple stakeholders, multiple groups providing requirements, and the level of criticality of the project is likely to result in the presence of both a greater number of environmental cues in general, and a higher proportion of heterogeneous cues. The presence of more and potentially conflicting cues will potentially reduce the ability of the team to respond to the environment.

Additionally, the geographic dispersion of the team places an additional level of communication overhead that affects the manner by which the environmental cues are propagated and processed by the entire team. When teams are collocated in the same room, face-to-face discussion and spontaneous and serendipitous communication naturally occur (Kiesler and Cummings 2002). Teams that are separated by distance

require additional formal communications protocols to ensure that important information is shared across all members of the team. In addition, geographic distribution has been found to increase the level of intra-team conflict (Hinds and Mortensen 2005).

The discussion above motivates the following hypothesis:

Hypothesis 2a: Structural complexity will negatively moderate the impact of agile methodologies on project success.

Technical complexity arises from the combination of scope and structure of the project system, and the ambiguity regarding the use of various technologies to complete the project (Shenhar and Dvir 1996). In this study, technical complexity is theorized as the combination of external integration requirements, the number of technology platforms, and the number of distinct modules that are contained within the project system. Because agile teams focus on the ability to make rapid adjustments throughout the project, the level of external integration required may hinder adaptation, either due to limitations of the remote system, the team's lack of authority to modify the external system, or due to the delivery schedule of the remote system (Meyer and Curley 1991). Integration with external systems is a form of reciprocal interdependence. Reciprocal interdependence means that the outputs of one element become the inputs of another, and is considered the highest level of interdependence-driven complexity (Thompson 2003). Finally, the number of platforms and number of functional modules of systems have been shown to cause an increase in technical complexity (McKeen et al. 1994; Meyer and Curley 1991).

High technical complexity impacts both the ability for agile teams to react to change in the environment, and the ability for project teams to evaluate the

requirements of the system and process those requirements into a design (Tait and Vessey 1988). As more features of the agile development project system require interaction with external systems, the impact of integration on the agile development process can grow more severe. Because most agile methodologies stress the practice of incremental design, early phases of the project may not recognize the pervasive nature of the system integration task. As more of the system is designed, a simple integration architecture that was appropriate for an earlier version of the system may need to be recreated to be robust enough for the full integration requirements of the system. Further, agile methodologies manage the risk of incremental design via the practice of continuous refactoring, and the supporting technologies of source code control and automated unit testing. However, projects teams with higher integration requirements may control less of the IT architecture that is relevant to the project, restricting the ability to perform extensive and efficient refactoring (Fowler 1999).

Even in the case that the existing system changes are possible, the iterative work process of agile development teams still may be impacted. Agile method philosophy dictates that the most valuable feature should be addressed first. However, if a very valuable feature depends upon a to-be enabled external integration, the agile team may be forced to delay the more valuable feature, and work on other, less valuable features first. Agile methodologies feedback loops provide important insight to the team about the congruence of the developed software with business requirements. Delaying feedback on important requirements may delay or reduce the ability to understand a broader set of project issues, thus reducing the ability of agile development to impact project success.

Finally, many legacy systems are production systems, and often do not have fully featured, open availability, test environments. Additionally, since external systems were often built before the adoption of automated testing methodologies, ad hoc and continuous testing against legacy systems often cannot be accomplished. This leads to the necessity to perform manual integration testing and may lead to the reduction in speed of the CI process. As discussed in Chapter 2, the pace of the CI process dictates the pace of deploying changes to the code base. A reduction in the pace of CI negatively impacts the ability to obtain feedback from the environment.

Based upon this discussion, the following is hypothesized:

Hypothesis 2b: Technical complexity will negatively moderate the impact of agile method use on project success.

The dynamic nature of the systems development environment is reflected in the extent to which the project team must respond to changing requirements throughout the duration of the project. Changing requirements are caused by ambiguity in initial requirements definition, as well as actual volatility in the system requirements due to emergent understanding of the business problem, or due to real organizational or technical change (McKeen et al. 1994; Meyer and Curley 1991; Ribbers and Schoo 2002). Environmental dynamism impacts project success because it potentially reduces the ability for the organization to possess the necessary information to make a decision (Thompson 2003), or impacts the continued relevance of previously defined requirements or courses of action. As changes occur within a project environment, uncertainty rises due to the reduction in the team's ability to clearly identify cause and effect relationships as to how environmental factors influence the situation, the

reduction in the ability to predict the success of decisions, and to confidently predict the likelihood of success of actions (Duncan 1972; Xia and Lee 2005). It is specifically because of this uncertainty that agile methodologies have developed their “sense and respond” nature (Lyytinen and Rose 2006).

Traditional plan-based development approaches generally perform poorly in dynamic environments while iterative and adaptive approaches fare worse in stable predictable environments (Eisenhardt and Tabrizi 1995; Port and Bui 2009). Although requirements change is an expected part of every software development effort (Schwaber 1996), some efforts are likely to have higher or lower levels of dynamism. Because dynamism has been identified as a key driver of software project failures, as defined by scope, time and budget constraints (Keil 1995), methodologies that better manage requirements changes should positively impact project performance.

However, when dynamism is low, it should be expected that the impacts of experiential or adaptive processes would be reduced. These processes add a measure of overhead to acquire feedback, adapt and respond. When this dynamism is not present, or when a problem is not novel, this overhead may simply be waste and add inefficiency. Therefore, agile method impacts would be expected to be higher in environments in which environmental dynamism is high:

Hypothesis 2c: Environmental dynamism will positively moderate the impact of agile methodologies on project success.

Control Variables

Project Size

Project size is expected to have an impact on project success, and the literature is inconsistent as to whether agile development methodologies are appropriate for larger projects. As projects grow in size, complexity increases, as does the potential for exceptions. Additionally, project size will have an impact on the length of time that it takes to build the software system. As a project continues across time, additional external factors of uncertainty such as changing business requirements and market conditions, may impact the success of the project.

Organization Size

Organization size has long been used as a proxy for a number of organizational characteristics, including structure, complexity, resource availability, among others. However, organizational size has been specifically linked to impacts on IS project implementation cost, time & deployment strategy (Mabert et al. 2003).

Time since Agile Adoption

Time is an important element in agile method use. First, agile practitioners recommend that teams adopt agile practices gradually. Rather than adopting a number of new practices, practitioners recommend that teams identify those practices that the believe may deliver the most impact first then add additional practices as their routines evolve. Over time, it is expected that teams will adopt more agile practices.

Additionally, as teams become more familiar with the practices, and integrate the philosophy of agile methodologies into their motivation, they become familiar with the

interplay between the practices, and can more effectively bring these practices to bear in specific situations.

Team Member Skill

Team member skill is measured as a control variable as it is likely to affect project performance in general regardless of method. Boehm & Turner (2004) claimed that team member skill was key to agile method performance, and more specifically that high skill members of the team are more important to the success of agile projects than traditional plan-based approaches. However, the various agile methodologies make conflicting claims and prescriptions regarding team member skill. FDD defines two levels of developer roles required in the team, chief programmer and class owner. While this makes an assumption about skill, the method doesn't clearly make statements regarding skill level. Scrum's three team roles do not mention skill level at all.

Team Member Agile Experience

Team member experience with agile most likely did not begin with the current project team. Because of this, if an organization that recently adopted agile acquired significantly experienced resources to implement its method and execute its project, team member agile experience is potentially a separate impact on project success.

Chapter Summary

Agile methodologies make a number of normative claims regarding the impacts of agile methodologies. In this chapter, a research model was described that tests the theoretical model presented in Chapter Two. The research model included several hypotheses about the relationships between the constructs, and are summarized in Table 3.2.

Table 3.2. Summary of Hypotheses

Agile Method Use
H1: The Extent of Agile Method Use will positively impact Project Success
Uncertainty
H2a: Structural Complexity will negatively moderate the impact of Agile Method Use on Project Success
H2b: Technical Complexity will negatively moderate the impact of Agile Method Use on Project Success
H2c: Environmental Dynamism will positively moderate the impact of Agile Method Use on Project Success

In Chapter Four, describe the research study that was performed to test the model presented in this chapter is described.

Chapter Four: Research Methodology

Introduction

The intent of this chapter is to develop and describe the research design that was utilized to complete this study. According to Creswell, a research design should illustrate how the sample, methodologies, and general study structure work together to answer the central research questions posed (Creswell 2003).

Chapter three identified the research questions, propositions and units of analysis for this study. This chapter will deal specifically with the logic linking the sample, methodologies, and anticipated data to the propositions, and the criteria for the evaluation of this data in respect to the propositions.

As described previously, the research on agile methodologies has been characterized by the measurement of particular agile method practices, such as pair programming and code ownership. As described in Chapter Three, agile method use is conceptualized as a multi-dimensional construct. Because of this new conceptualization of agile method use, this research was exploratory in nature. The research was conducted by developing a new quantitative survey instrument, which was used to test the propositions described in chapter three.

Construct Operationalization

Whenever possible and applicable, items were adapted from previously existing questionnaires. However, because some of the constructs proposed in this study are new, additional items have been created for this study. Additionally, several additional dimensions were added to existing constructs.

Because only team members could reasonably be expected to accurately answer all of the questions on the questionnaire, respondents were first asked what their role on the project was. Based upon this initial response, the respondent would be presented with the appropriate sections of the survey. A listing of the sections of the survey presented to each class of respondent is included in Table 4.1

Table 4.1. Sections of Questionnaire Presented to Each Respondent Type

Section of Survey	Team	IT Mgmt	Stakeholder
Agile Method Use	X		
Technical Complexity	X		
Structural Complexity	X	X	
Requirements Dynamism	X	X	X
External Dynamism	X	X	X
Project Success	X	X	X

Extent of Agile Method Use

Agile method use was measured by creating a set of items that indicated the agile practices that are both universal to agile methodologies, and likely to directly impact project success The agile practices measured were: Reduced Up Front Planning, iterative delivery, customer feedback and technical feedback. This portion of the questionnaire was answered only by the team members involved in the day to day execution of the project.

For each of these processes, several items were developed utilizing both the agile manifesto itself, as well as agile method practitioner explanatory literature (Fowler and Highsmith 2001; Highsmith 2002). Each agile principle was measured with a seven-point scale ranging from *Strongly Disagree* to *Strongly Agree*. Respondents were given the option to also select *"Don't Know"*. The agile feedback processes were measured as described in Table 4.2.

Table 4.2. Items Indicating the Extent of Agile Method Use

Reduced Up Front Planning
“Please indicate the extent to which you agree or disagree that the following statements ACCURATELY REFLECT the project team's BEHAVIOR during the project"

1. The team spent less than 10% of the total project timeline on up-front planning (planning that occurred before ANY coding began).
2. At the beginning of the project, the team tried to make only the decisions that were necessary for coding to begin.

Iterative Delivery
“Please indicate the extent to which you agree or disagree that the following statements ACCURATELY REFLECT the project team's BEHAVIOR during the project"

1. The team executed development using a series of short cycles or iterations.
2. At the beginning of each development cycle, the team and business owners agreed on what would be delivered during the development cycle.
3. At the end of every development cycle, the code was deployable.

Environmental Feedback
“Please indicate the extent to which you agree or disagree that the following statements ACCURATELY REFLECT the project team's BEHAVIOR during the project"
The team had a short meeting every day to discuss what was going on that day.

1. On a regular basis, the team demonstrated working software to the customer/user.
2. The team had a review/verification meeting with stakeholders to demonstrate when software features were complete.
3. On a regular basis, the team reflected on previous work, and looked for ways to improve team performance.
4. The team had a short meeting every day to discuss what was going on that day.

Technical Feedback
“Please indicate the extent to which you agree or disagree that the following statements ACCURATELY REFLECT the project team's BEHAVIOR during the project"

1. Members of the team integrated code changes as soon as possible.
2. Every programmer was responsible for writing automated tests for the code he or she wrote.
3. Programmers ran a set of automated tests until they all ran successfully before checking in changes.

Scale:
5-point Likert scale: Strongly Disagree to Strongly Agree [or Don’t Know]

Project Success

Because agile methodologies stress the development of shared understanding amongst the team and stakeholders, the questions about the success of the project were presented to all groups of respondents. Project success was measured by three constructs: (a) Project Management Success, (b) Product Quality, and (c) Perceived Project Impacts. Project management success will be measured using the "iron triangle" measures of budget forecast, scope forecast and duration forecast. Previous literature indicates that these measures are subject to gaming based upon political pressures. Even so, most IS projects, whether utilizing agile method or not, are expected to be justified based upon some set of up-front cost estimates, and the ability to deliver a suitable system within the constraints of a business-driven timeline.

Table 4.3. Items Indicating Project Management Success

Project Management Metrics 1. In comparison to the initial budget estimate the final budget was: 2. In comparison to the initial time estimate the final project duration was: 3. In comparison to the initial feature set (scope) the final project scope was: Scale 5-point likert scales: 1. "Very much lower" to "Very much higher" 2. "Very much shorter" to "Very much longer" 3. "Very much smaller" to "Very much larger"

Even though significant criticism has been leveled at agile method for its lack of up front planning, agile method have been reported to deliver projects within the high level estimates generated at the beginning of projects (Coad et al. 1999; Highsmith 2002). For these reasons, it is reasonable to measure the "iron triangle" as an indicator

of agile project success. The measures for project management success are listed in Table 4.3.

Project management success was measured with three constructs: (a) budget estimation, (b) timeline estimation, and (c) scope estimation. Each construct will be measured with six items. The items for each construct are very similar, and utilize the a five point likert scale ranging from *Very Much Lower* to *Very Much Higher*. In addition, the respondents were able to indicate “don’t know”, or that there was not an initial estimate made.

Product quality was measured with a set of scales adapted and modified from Wixom and Todd (2005). Because these original scales were specific to particular technology contexts, they were generalized. The product quality dimensions measured were: (a) quality, (b) usefulness, (c) completeness, (d) reliability, (e) accuracy, and (f) suitability. The items used for Product quality are presented in Table 4.4.

Table 4.4. Items Indicating Product Quality

Prompt
Please give your opinion about the following statements about the project system:
Quality

1. In terms of system quality, I would rate [the project system] highly.
2. Overall, [the project system] is of high quality.
3. I would give the quality of [the project system] a high rating.

Usefulness

1. [The project system] improves users abilities to perform their tasks.
2. The project system] allows users to get work done more effectively.
3. [The project system] allows users to get their tasks done more quickly.

Completeness

1. [The project system] provides users with a complete set features and information.
2. [The project system] is a comprehensive solution.
3. [The project system] provides users with all needed information to do their tasks in the system.

Reliability

1. [The project system] operates reliably.
2. The company can rely on [the project system].
3. The operation of [the project system] is dependable.

Accuracy

1. [The project system] properly performs the tasks it was intended to perform.
2. There are few errors or bugs in [the project system].
3. The information provided by [the project system] is accurate.

Suitability

1. The [the project system] delivered the desired project outcome.
2. The [the project system] accomplishes what was needed.
3. The [the project system] does what was it is supposed to.

Scale
5-point likert scale: Definitely Not True to Definitely True

The perceived benefits of the system were measured with two constructs asking respondent their perceptions of the organizational benefits associated with the project. These measures are summarized in Table 4.5.

Table 4.5. Items Indicating Organization Benefits

Customer Satisfaction
Prompt
Please give your opinion about these statements related to YOUR PERCEPTION of satisfaction with the project system:

1. The users are satisfied with [the project system].
2. The customer is satisfied with [the project system].
3. [The project system] satisfies the company's needs.

Organizational Impacts
Prompt
Please give your opinion about these statements related to YOUR PERCEPTION of satisfaction with the project system:

1. Because of this project, our organization can better realize its goals.
2. This project helped our organization to perform better.
3. Our organization is more competitive because of this project.
4. This project was worth the investment.
5. This project delivered on its promise.

Scale
5-point likert scale: Definitely Not True to Definitely True

Control Variables

Six control variables were measured: project size, organization size, management support, team member experience, team member agile experience, and time since the organization adopted agile method. Management support was measured through modifying items previously used in Purvis, et al. (Purvis et al. 2001), with slight modifications.

Organization size was measured by asking the respondent to estimate the total number of people in the organization, while team size was measured by asking the respondent to estimate the number of people who participated on the team.

Experience level was measured by asking the respondent how long he had been working in software development. Agile experience was measured by asking the

respondent how long she had been working on agile projects in general. Finally, each respondent was asked to indicate how much time had elapsed since the organization had adopted agile methodologies. Control variable items can be found in the full survey instrument in Appendix A.

Pre-Testing the Survey

The lack of pre-testing has often been cited as a major source of failures of otherwise well-conceived studies. Converse & Presser (Converse and Presser 1986) argue that even well-established item scales should not be assumed to properly translate into new contexts. Because of the novel nature of the items in this survey, and because the items utilized from previous research were modified, a two-stage confirmatory pre-test design was utilized.

The initial pre-test of the survey instruments was carried out using a convenient sample of five volunteers who were experts in agile development. Each volunteer was given the survey via the Qualtrics online survey tool which was used for the full survey implementation. As each volunteer completed the survey, he or she was asked to provide initial feedback as to the instrument, flow, item validity and applicability to agile method. Then, the volunteer was asked specific questions about the meaning of the indicator items on the survey. Some items were found to be ambiguous based upon the respondent's misidentification of the question's meaning and/or scale. Based upon the pretest, items were modified, flow was altered, and the final survey was created.

After the initial pre-test, a set of pilot tests with three groups of 8-15 respondents (36 total responses) was performed to test initial construct reliability. After each pilot

test, the group of respondents was asked about what they were thinking during each section of the survey, about sequencing and clarity of questions, and about areas of importance that were excluded from the survey. After this phase, the final version of the survey was produced.

Sample Design and Data Collection

A field survey of information systems development teams who self-identified as using agile methodologies was performed using multiple respondent design. We utilized a purposive sample design to recruit these teams. It is appropriate to administer this survey to a purposive sample, because all project teams do not use agile method. As the major constructs of interest involve the adoption of agile methodologies, sampling from outside the population active agile development teams would be inappropriate.

The sample was obtained via a two-phase recruitment effort. In the first phase, a passive invitation was addressed to two groups of self-reported agile practitioners. First, an email and web posting announcing the survey and calling for volunteers was posted to the membership list of a large agile affinity group in the Midwest. In addition, the invitation was posted by several recognized agile authorities on a set of agile affinity groups on the LinkedIn.com social networking platform. To attempt to generate a high response rate, the invitation and survey instrument included strong assurances of confidentiality and anonymity. Respondents were asked to identify a project that he or she worked on in the past year that utilized agile method, and the role that was played on the project. Additionally, the respondent was asked to give the time frame of when this project was completed.

Because the study sought to receive at least two respondents per team, respondents from these invitations were asked to provide additional contact information for other team members, IT management, and stakeholders. This phase of the data gathering generated about 75 responses of which only 22 were usable, and did not generate additional usable additional team responses.

The second phase of data gathering involved direct recruitment of teams. The researcher solicited organizations to volunteer by presenting the research project at several agile affinity groups in the Midwest, and by directly contacting organizations known to practice agile methodologies. Once an organization expressed interest, a short explanation of the research project was shared, and any necessary corporate approval for the research project was obtained. Two incentives were provided during this phase. First, individuals were offered a $5 Starbucks gift card for fully completing the survey. Additionally, respondents were given the opportunity to request a white paper detailing results of the study. Finally, organizations that provided 30 or more respondents from 5 or more teams were given the added incentive of a customized report that contrasted the results of the organization's teams with the full sample. Because teams were identified by the organization, each team was given a unique identifier which was sent to the organization to distribute. Based upon these unique identifiers, responses were able to be tied to particular teams.

Table 4.6 describes the structure of the sample acquired. Because the composition of agile teams is dependent upon organization, an "other" category was added to the survey. Where it was clear that the respondent should have chosen one of

the defined options, the record was recoded. The recoded numbers are listed in the table.

Table 4.6. Demographics Breakdown of Respondents

Role	Phase 1	Phase 2	Total	%	Cumulative %
Business Stakeholder	3	42	45	11.4%	12.2%
IT Management	4	27	31	8.4%	20.6%
Team Lead	4	35	39	8.9%	31.2%
Architect	2	4	6	1.9%	32.8%
Developer	3	126	129	34.1%	67.8%
Project Manager	5	14	19	3.8%	72.9%
QA/Testing	0	32	32	8.9%	81.6%
Business Analyst	0	23	23	6.2%	87.8%
Other	1	44	45	16.3%	100.0%
Totals	22	347	369		

Based upon the role selected, each user was presented with the sections of the survey noted in Table 4.7. Respondents who chose the “Business Stakeholder” role were presented with the stakeholder section of the survey. Respondents who chose the “IT Management” role received the IT Management sections of the survey. All other roles received the full team member survey.

Response volume from the project teams varied, with project response rates ranging from 1 to 20 respondents. The full breakdown of team responses is presented in Appendix B. Responses were received from 83 teams, of which 57 teams provided more than one response.

Because all of the survey items were not presented to each class of respondent, there are several constructs with data missing for particular classes. Table 4.7 presents a summary of the missing data by item. There are two classes of “missing” data. Missing (not presented) indicates the number of respondents who did not have the opportunity to provide data because the question was not presented. The missing

column describes true “missing” data, meaning that the item was presented to the respondent, but no answer was provided.

Multiple imputation was performed on the data, using 10 imputations. After this process, there were still four records for which data could not be imputed. Those four cases were dropped, reducing the size of the sample to 369 respondents.

Table 4.7. Missing Data Summary for 373 Cases

Variable	Construct	Non-Missing	Missing (not presented)	Missing
Team Size	Structural Complexity	325	42(11%)	6(1.6%)
Team Companies	Structural Complexity	326	42(11%)	5(1.3%)
Team Departments	Structural Complexity	326	42(11%)	5(1.3%)
Project User Groups	Structural Complexity	325	42(11%)	6(1.6%)
Geographic Distance	Structural Complexity	326	42(11%)	5(1.3%)
Project Criticality 1	Structural Complexity	325	42(11%)	6(1.6%)
Project Criticality 2	Structural Complexity	325	42(11%)	6(1.6%)
Project Criticality 3	Structural Complexity	325	42(11%)	6(1.6%)
Tech Comp 1	Technical Complexity	293	73(19.6%)	7(1.8%)
Tech Comp 2	Technical Complexity	293	73(19.6%)	7(1.8%)
Tech Comp 3	Technical Complexity	293	73(19.6%)	7(1.8%)
Tech Comp 4	Technical Complexity	293	73(19.6%)	7(1.8%)
Req. Dynamism 1	Dynamism	362	0	11(2.9%)
Req. Dynamism 2	Dynamism	361	0	12(3.2%)
Req. Dynamism 3	Dynamism	362	0	11(2.9%)
Ext. Dynamism 1	Dynamism	362	0	11(2.9%)
Ext. Dynamism 2	Dynamism	361	0	12(3.2%)
Ext. Dynamism 3	Dynamism	361	0	12(3.2%)
Tech. Dynamism 1	Dynamism	362	0	11(2.9%)
Tech. Dynamism 2	Dynamism	361	0	12(3.2%)
Reduced Up Front 1	Agile Use	289	73(19.6%)	11(2.9%)
Reduced Up Front 2	Agile Use	289	73(19.6%)	11(2.9%)
Env Feedback 1	Agile Use	290	73(19.6%)	10(2.6%)
Env Feedback 2	Agile Use	290	73(19.6%)	10(2.6%)
Env. Feedback 3	Agile Use	290	73(19.6%)	10(2.6%)
Env. Feedback 4	Agile Use	290	73(19.6%)	10(2.6%)
Env. Feedback 5	Agile Use	290	73(19.6%)	10(2.6%)
Tech Feedback 1	Agile Use	290	73(19.6%)	10(2.6%)
Tech Feedback 2	Agile Use	290	73(19.6%)	10(2.6%)
Budget Outcome	Budget Outcome	359	0	14(3.8%)
Time Outcome	Time Outcome	360	0	13(3.5%)
Scope Outcome	Scope Outcome	359	0	14(3.8%)
Reliability 1	Quality	357	0	16(4.3%)
Reliability 2	Quality	357	0	16(4.3%)
Reliability 3	Quality	357	0	16(4.3%)
Usefulness 1	Quality	360	0	13(3.5%)
Usefulness 2	Quality	359	0	14(3.8%)
Usefulness 3	Quality	360	0	13(3.5%)

Table 4.7. (cont'd)

Variable	Construct	Non-Missing	Missing (not presented)	Missing
Completeness 1	Quality	356	0	17(4.6%)
Completeness 2	Quality	357	0	16(4.3%)
Completeness 3	Quality	356	0	17(4.6%)
Benefits 1	Org. Benefits	358	0	15(4.0%)
Benefits 2	Org. Benefits	357	0	16(4.3%)
Benefits 3	Org. Benefits	358	0	15(4.0%)

Chapter Summary

This chapter described the method used for the measurement and collection of data for this study on the use and impacts of agile methodologies. Chapter five describes the results.

Chapter 5: Analysis and Results

Introduction

This chapter presents results of statistical analysis performed to test the specified hypotheses on the data obtained from the research described earlier.

Analysis Approach

Most of the measures used in this study were either new or were modified from their original sources, so the analysis performed was exploratory. First the instrument was verified utilizing exploratory factor analysis. Next the nature of the relationships between the factored constructs was investigated. Finally a test of the full model was performed using a generalized linear model. The results of the analysis are described below.

In Chapters 2 and 3, it was theorized that Agile Method Use would positively impact project success. Further, it was proposed that this relationship was moderated by three components of uncertainty: structural complexity, technical complexity, and dynamism. The independent variable of Extent of Agile Method Use first.

Extent of Agile Method Use

Because the proposed theoretical model and the items developed to indicate the constructs were new, a exploratory factor analysis was performed. When using a new measurement instrument, it likely that some items will not perform as well as others, and exploratory factor analysis can be used to identify poorly clustered items. All exploratory factor analyses in this chapter were performed using Stata, utilizing principal components factors, and oblique oblimin rotation. Using oblique rotation rather than an orthogonal rotation is appropriate in this case as the factors we are analyzing are

dimensions of a higher-order construct, and would be expected to correlate with one another. Agile method use was originally predicted to include four dimensions, however the data factored into three factors, environmental feedback, technical feedback, and reduced up front planning. The theorized factor of iterative delivery did not materialize in this data set and was dropped from the analysis, although one of the items that was anticipated to load on iterative delivery loaded instead on environmental feedback. Factor loadings are presented in Table 5.1

As can be seen all with loadings are over .70, and no cross loadings over .30. Descriptive statistics for the items included in the factors are presented in Table 5.2.

Table 5.1. Factor Loadings for the Dimensions of Agile Method Use

Item	Env. Feedback	Technical Feedback	Reduced Up Front Planning
Less than 10% Upfront planning	-.0054	.0593	.8617
Minimum decisions before beginning work	.0144	-.0256	.8964
Short Cycles	.7003	.0224	.2028
Defined Scope for Cycles	.8656	-.1686	.0384
Daily Feedback Meeting	.7142	.1511	-.0606
Regular Demonstration	.7299	.1980	-.0054
Verification Meetings	.8138	.0120	-.0738
Tests Required	.0145	.9175	.0590
Tests Run Before Check In	.0141	.9547	-.0285

Environmental feedback is represented by those items that indicated the team's use of structures and processes that provided opportunities to collect and process environmental data. As discussed in Chapter Two, the ability to regularly allow feedback to cross the team buffer is an important contributing factor to team adaptability, and performance in an uncertain environment (Burke et al. 2006; Eisenhardt and Tabrizi

1995; MacCormack 2001). Technical Feedback was indicated by the process of automated testing – both the practice of requiring tests and requiring their successful execution before checking in changes.

Reduced up front planning reduces waste regarding the definition of requirements and actions that will be taken in the future, and are likely to change, and decreases the time between project commencement and initial feedback. Reduced up front planning was indicated by two items that related directly to minimizing the time and decisions made before starting work.

Table 5.2. Descriptive Statistics for Agile Use Items

Item	Mean	Std. Dev.	Min	Max
Less than 10% Upfront planning	3.13	1.10	1	5
Minimum decisions before beginning work	2.98	1.06	1	5
Defined Scope for Cycles	3.74	.99	1	5
Daily Feedback Meeting	4.23	.97	1	5
Verification Meetings	4.01	.85	1	5
Tests Required	3.47	1.30	1	5
Tests Run Before Check In	3.56	1.23	1	5

Outcome Variables

The outcome variables for this study were quality, organizational benefits, and project management outcomes. We measured quality with six dimensions, perceived quality, reliability, usefulness, accuracy, completeness, and suitability. In initial exploratory factor analysis, these dimensions loaded onto three factors, with usefulness and completeness loading on their own, and the remaining four dimensions loading together. Because Chapter Three proposed perceived quality to be indicated by the other five factors, we dropped it to determine if it influenced the other factors to load

together. However, after dropping the perceived quality items, the data still loaded on three factors.

Accuracy did not load on a single factor, and was dropped. Reliability and perceived quality loaded together on a single factor. This combined factor was compared with the quality and reliability factors separately, and was found to perform nearly identically in correlation/covariance comparisons, and in simple regression analyses. The combined factor performed worse than the reliability-only construct, and thus, perceived quality was dropped.

Table 5.3. Factor Loading for Product Quality and Benefits Items

Item	Reliability	Usefulness	Complete-ness	Org. Benefits
System operates reliably	.9476	.0115	.0250	-.0587
Company can rely on it	.9255	-.0046	.0173	.0541
It is dependable	.9537	-.0070	-.0310	.0348
Improves users' tasks	.0516	.8981	-.0246	.0147
Helps users do tasks	.0268	.9375	.0298	-.0496
Users get work done	-.0769	.9010	-.0059	.0807
Complete set of features	.0043	-.0435	.8800	.0550
All needed information	-.0398	-.0353	.9120	.0652
Comprehensive solution	.0839	.1205	.8115	-.1250
Org realize goals	.0534	.0386	-.0007	.8560
Org performs better	-.0064	.0730	.1104	.7955
Org more competitive	.0018	-.0380	-.0468	.8844

In the interest of parsimony, reliability items loaded at the highest level were retained while accuracy and suitability items were dropped, leaving Quality with three factors. Then, an additional EFA added the organizational benefits items. Three of the

organizational benefits loaded on one factor and were retained; the final EFA for the product quality dimensions and org benefits factor is listed in Table 5.3.

Table 5.4. Descriptive Statistics for Product Quality and Benefits Items

Item	Mean	Std. Dev.	Min	Max
System operates reliably.	4.20	.739	1	5
Company can rely on system	4.30	.661	1	5
The system is dependable	4.26	.694	1	5
System improves users' tasks	4.33	.709	1	5
Helps users complete tasks	4.28	.752	1	5
Helps users get work done	4.18	.782	1	5
Complete set of features	3.82	.918	1	5
All needed information	3.61	1.09	1	5
Comprehensive solution	3.73	.957	1	5
Org can better realize goals	4.08	.779	1	5
Org performs better	3.96	.872	1	5
Org more competitive	4.01	.860	1	5

The three retained quality dimensions are reliability, usefulness, and completeness. Reliability includes the items that relate to the sense that the system will not let the user or the company down, in short, the level to which the organization can depend on the system. Usefulness includes those items that relate to the system's ability to help uses improve the manner in which they complete their tasks, get their tasks done at all, and complete their work. Completeness was indicated by the three items that denoted the system's features as being complete, information as being complete, and solving a complete business problem. The descriptive statistics for the product quality and benefits items are listed in Table 5.4.

Uncertainty

The discussion in Chapter Two theorized that the factors of Structural Complexity, Technical Complexity, and Environmental Dynamism indicate uncertainty in a software development project. This theorized factor model was tested with exploratory factor analysis, and it was found that the data loaded into seven dimensions of uncertainty, three that related to dynamism (rather than the proposed two), and two that related to technical complexity.

Dynamism

It was initially proposed that dynamism would come from three sources, changes in requirements, changes in the technical environment, and changes to the organizational environment outside of the team. Dynamism did load into these three factors, with a minimum loading of .6924, with no cross loadings over .23. Factor loadings for the dimensions of dynamism are presented in Table 5.5, and descriptive statistics are listed in Table 5.6. External dynamism is indicated by items that describe changes that the project was associated with that occurred outside the project. These changes included associated business processes, organizational structure and information needs. Requirements dynamism relates to changes in actual system requirements during the project. Finally, technical dynamism relates to changes in infrastructure and software development environments during the project.

Table 5.5. Factor Loadings for the Dimensions of Dynamism

Item	Requirements Dynamism	Technical Dynamism	External Dynamism
Early Req. Change	.7815	-.0220	-.0033
Late Req. Change	.8252	-.0352	.0596
Final Req. Different	.8280	.0816	-.0359
IT infrastructure change	-.0145	.8721	.0637
SW Dev. tool change	.0464	.8918	-.0534
Business Process Change	-.0183	-.1275	.8849
Org Structure Change	-.1110	.2814	.6973
Information Needs Changed	.1999	.0449	.7235

Table 5.6. Descriptive Statistics for Uncertainty Items

Item	Mean	Std. Dev.	Min	Max
Early Req. Change	3.68	.973	1	5
Late Req. Change	3.20	1.02	1	5
Final Req. Different	3.08	.996	1	5
IT infrastructure change	2.22	.907	1	5
SW Dev. tool change	2.16	.884	1	5
Business Process Change	2.93	1.08	1	5
Org Structure Change	2.27	.865	1	5
Information Needs Changed	2.49	.948	1	5

Complexity

In Chapter Two, it was theorized technical complexity to be indicated by two factors, structural complexity and technical complexity. In the study data, the technical complexity items loaded into two dimensions: Integration complexity and system complexity. Integration complexity consists of the items related to the number of software environments and platforms used within the product system, as well as integration with external systems that was necessary in the project. System complexity

consists of the evaluation by the project team of the high-level project system complexity and organizational systems environment complexity. Factor Loadings for Technical Complexity are displayed in Table 5.7.

Table 5.7. Factor Loadings for the Dimensions of Technical Complexity

Item	Integration Complexity	System Complexity
Multiple SW Environments	.7043	.0102
Multiple Tech Platforms	.7408	-.0756
External Integration	.8133	.0083
Complexity of Integration	.7651	.0541
Org. System Environment Complex	.0710	.8673
Project System Complex	-.0604	.9031

Structural complexity was constructed as an index that averaged the items that indicated the size of the team, the number of departments represented on the team, the number of user groups providing requirements, the project criticality, and the geographic distribution of the team. Descriptive statistics of all complexity items are listed in Table 5.8.

Table 5.8. Descriptive Statistics for Technical and Structural Complexity Items

Item	Mean	Std. Dev.	Min	Max
Project part of strategic plan	4.34	.676	1	5
Project in response to competition	3.60	1.09	1	5
Project failure has financial impact	3.25	1.03	1	5
Org. System Environment Complex	2.61	1.11	1	5
Project System Complex	2.57	1.10	1	5
Multiple SW Environments	3.77	1.06	1	5
Multiple Tech Platforms	3.77	1.03	1	5
External Integration	3.77	1.03	1	5
Level of Integration Complexity	3.34	1.06	1	5

Once the first order factors were generated, a second-order EFA was performed by factor. The factor loadings for the theorized second order factors are presented in Tables 5.9a-d.

As can be seen in Table 5.9a and Table 5.9b, second order loadings of Quality and technical complexity were acceptable, with all loadings over .7.

Table 5.9a. Second Order Factor Loadings: Quality

Factor	Quality
Reliability	.7714
Usefulness	.7372
Completeness	.7783

Table 5.9b. Second Order Factor Loadings: Technical Complexity

Factor	Complexity
System Complexity	.7549
Integration Complexity	.7549

Table 5.9c presents the factor loadings for the Agile Method Use factor. Environmental feedback and technical feedback meet the threshold of .7 that indicate

suitable fit. Reduced up front planning loads only at the .69 level. This indicates only marginal fit, but due to fact that this scale is newly developed, .69 should be considered acceptable, although further development and improvement of this scale is required in future studies.

Table 5.9c. Second Order Factor Loadings: Agile Method Use

Factor	Agile
Environmental Feedback	.6963
Technical Feedback	.7289
Reduced Up-Front Planning	.6913

Table 5.9d presents the second order factor loadings for Dynamism. External and Technical dynamism each load above .7, but requirements dynamism loads only at .63. While this loading is low, this factor is conceptually a component of dynamism. Further, while a loading of .7 is generally considered the cutoff for suitable fit, when correcting for sample size, .6 may be considered acceptable in this case, with the sample size being higher than 300 cases.

Table 5.9d. Second Order Factor Loadings: Dynamism

Factor	Dynamism
Requirements Dynamism	.6252
External Dynamism	.7235
Technical Dynamism	.7253

Table 5.10. Correlations and Variance Inflation Factors

Factor	Quality	Agile	Structural Complexity	Technical Complexity	Dynamism	Management Support
Quality	1.05					
Agile	.1632 (.0003)	1.27				
Structural Complexity	-.0364 (.4858)	.2381 (.0000)	1.16			
Technical Complexity	.0270 (.6057)	.2343 (.0000)	.3003 (.0000)	1.18		
Dynamism	-.1178 (.0238)	.0038 (.9416)	.2012 (.0001)	.1594 (.0022)	1.07	
Management Support	.1339 (.0101)	.3810 (.0000)	.1017 (.0513)	.1631 (.0017)	-.0345 (.5091)	1.19

(Numbers in parentheses indicate p-value, Numbers on the diagonal represent VIF)

The correlations between the proposed model factors are shown in Table 5.10. Because of the highly correlated nature of several of the predictors a diagnostic test for multicollinearity was run, and the variance inflation factors are included on the diagonal of Table 5.10. The presence of a VIF over 10 is indicative of multicollinearity. As the highest VIF for the factors is 1.27, this indicates that multicollinearity is not an issue.

However, as can be seen the correlations between the theorized predictors and outcome variables are generally non-significant. This indicates that the data factors are either truly not correlated, or are being influenced by violations of the assumptions of correlation analysis. Correlation analysis depends upon the assumptions of linearity in the relationship between the two variables, normality of the distribution of each of the variables, and homoscedascity across the range of the relationship.

In the next section the analysis of the data as to whether it meets these and other assumptions of statistical analysis.

Analysis of Data Characteristics

Several tests were run to evaluate the conformance of the data to statistical assumptions. The description of the tests, results, and the implications of these tests are described below.

Tests for Normality

First, Shapiro-Wilk tests were performed on all of the model variables. Results are shown in Table 5.11. In this test, the null hypothesis that the data is normally distributed is rejected if the p value is less than .05. As shown in the table, all of the outcome variables, and the agile method use variable are below the .05 level, which

means the null hypothesis that they are distributed normally can be rejected. The three moderators in the model, Structural Complexity, Technical Complexity, and Dynamism all are above the .05 level, indicating that we cannot reject that they are distributed normally.

Table 5.11. Shapiro-Wilk Test for Normality of Distribution

Factor	p-value
Quality	.0007
Organization Benefits	.0000
Budget Outcome	.0012
Time Outcome	.0000
Scope Outcome	.0000
Agile Method Use	.0000
Structural Complexity	.4800
Technical Complexity	.8105
Dynamism	.0569

While the Shapiro-Wilk test can indicate whether or not the distribution may be normal, it does not indicate the nature of the non-normality. In order to understand the nature of the data distributions, a series of inspections of the data were performed, using histograms, box plots, symmetry plots and Q-Q normal plots. The output of these tests is shown in Figures 5.1-5.5

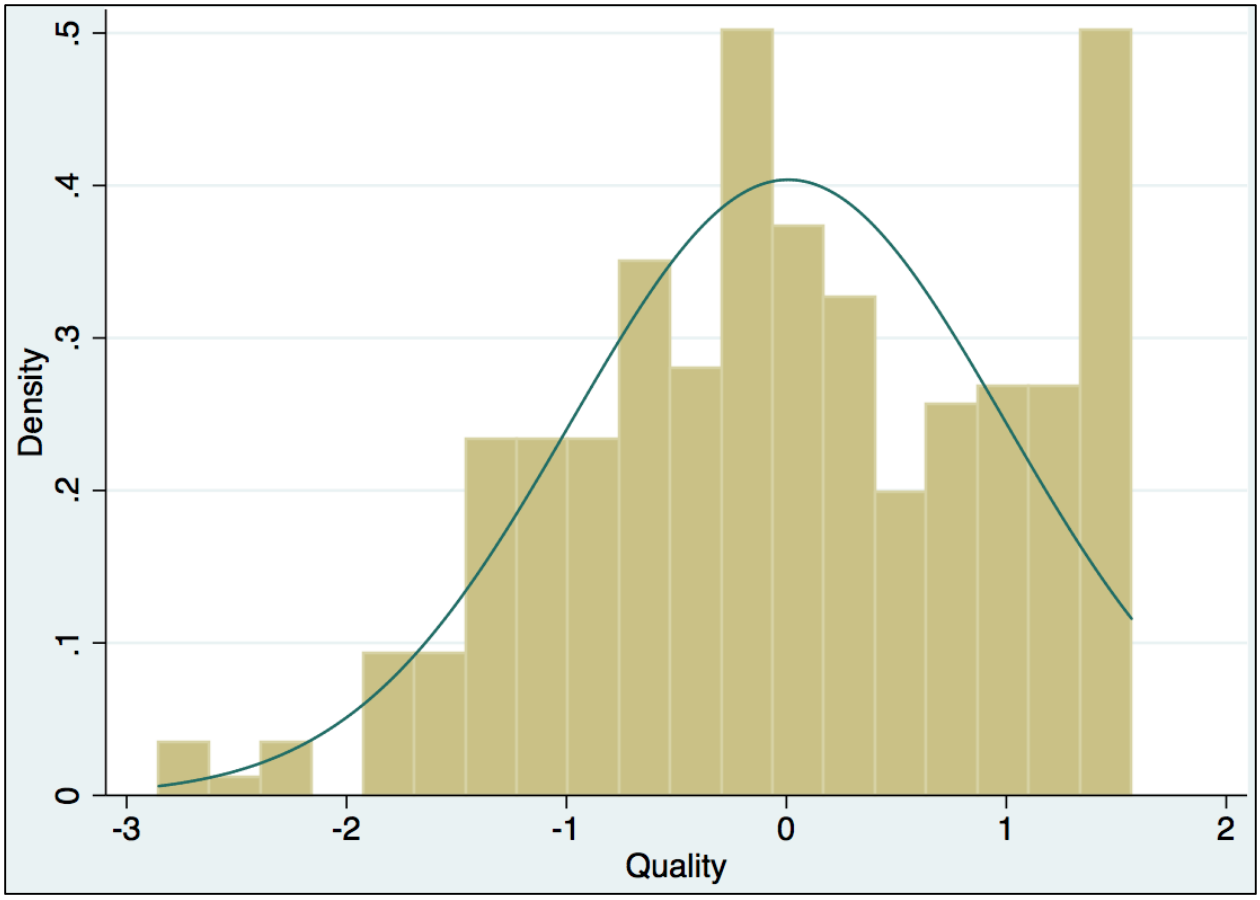

Figure 5.1a: Diagnostic Outputs for Quality - Histogram

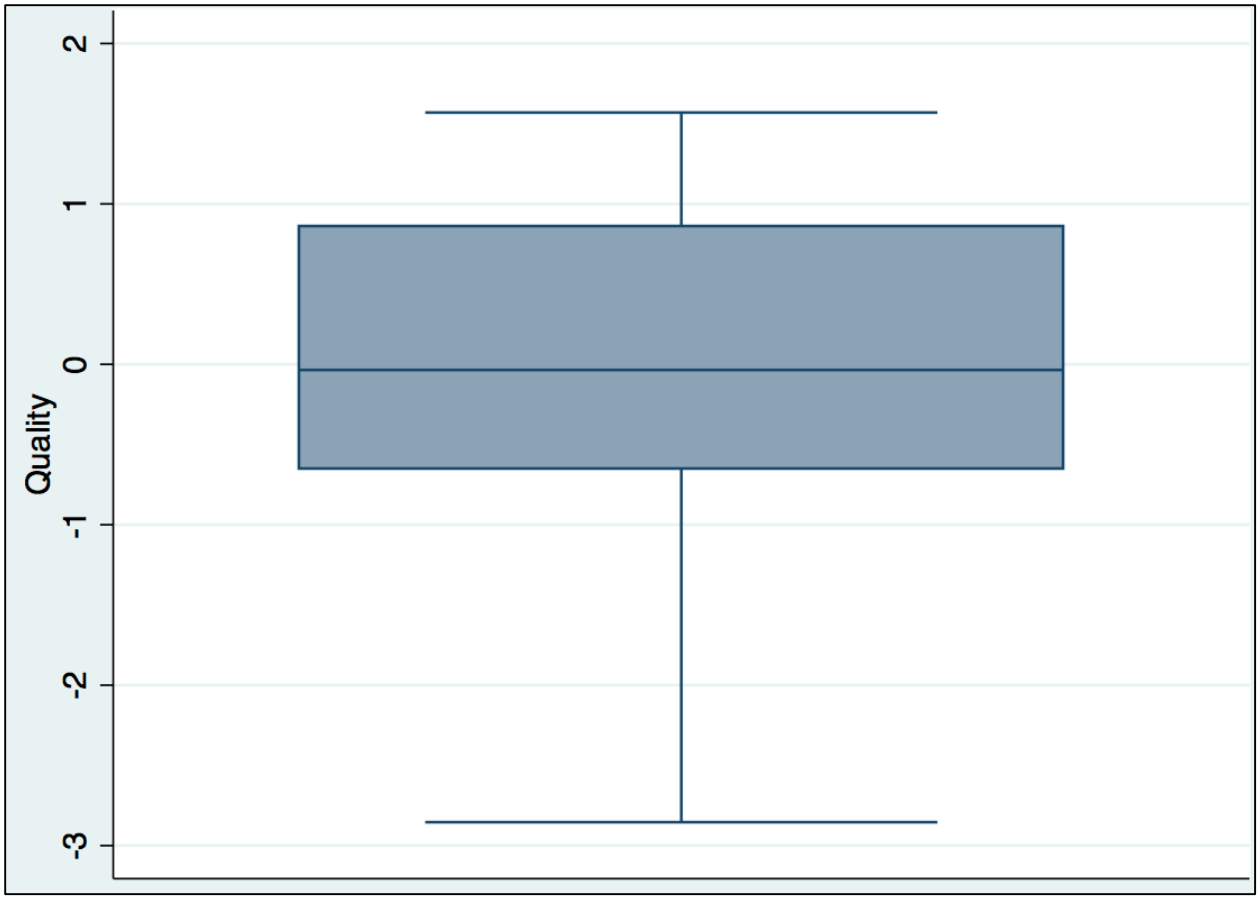

Figure 5.1b: Diagnostic Outputs for Quality – Box Plot

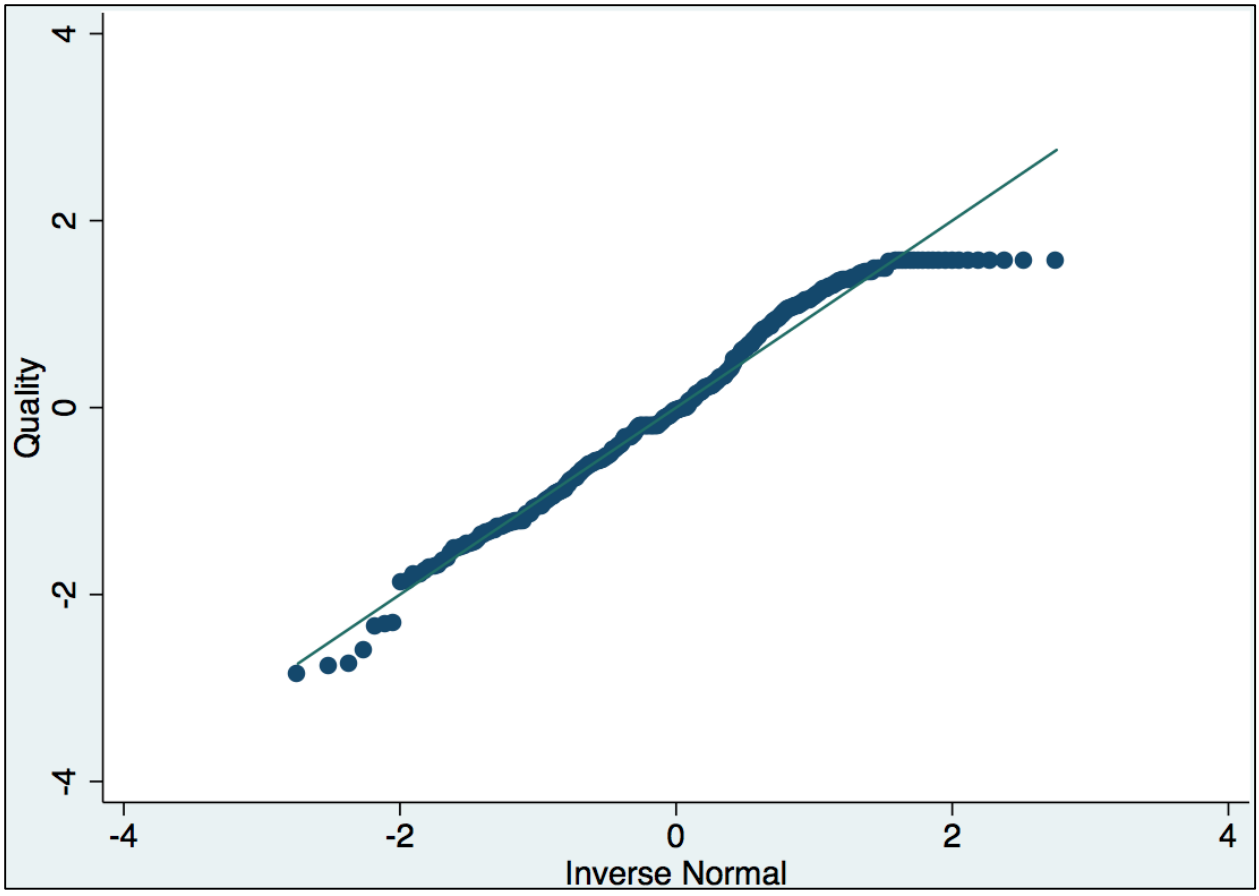

Figure 5.1c: Diagnostic Outputs for Quality – Q-Normal Plot

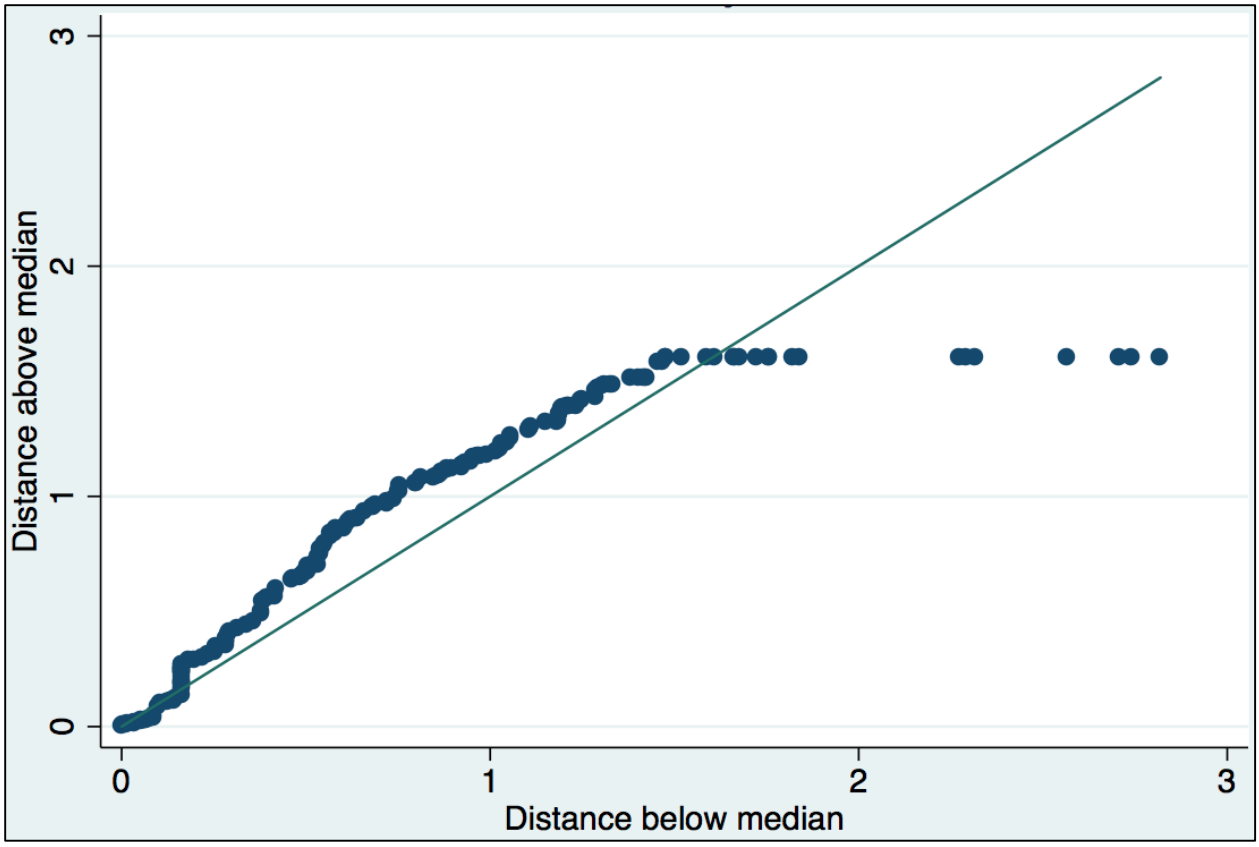

Figure 5.1d: Diagnostic Outputs for Quality – Symmetry Plot

As can be seen from Figures 5.1a and b, while no outliers are evident in the box plot, quality has a slight left skew. These tests confirm the results of the Shapiro-Wilk test, and give the additional insight that the right side of the distribution is heavy. The symmetry and q-normal plots (figures 5.1c and 5.1d) indicate that the right tail is the most problematic departure from normality.

Figures 5.2a-d display the output from the tests for normality for agile method use.

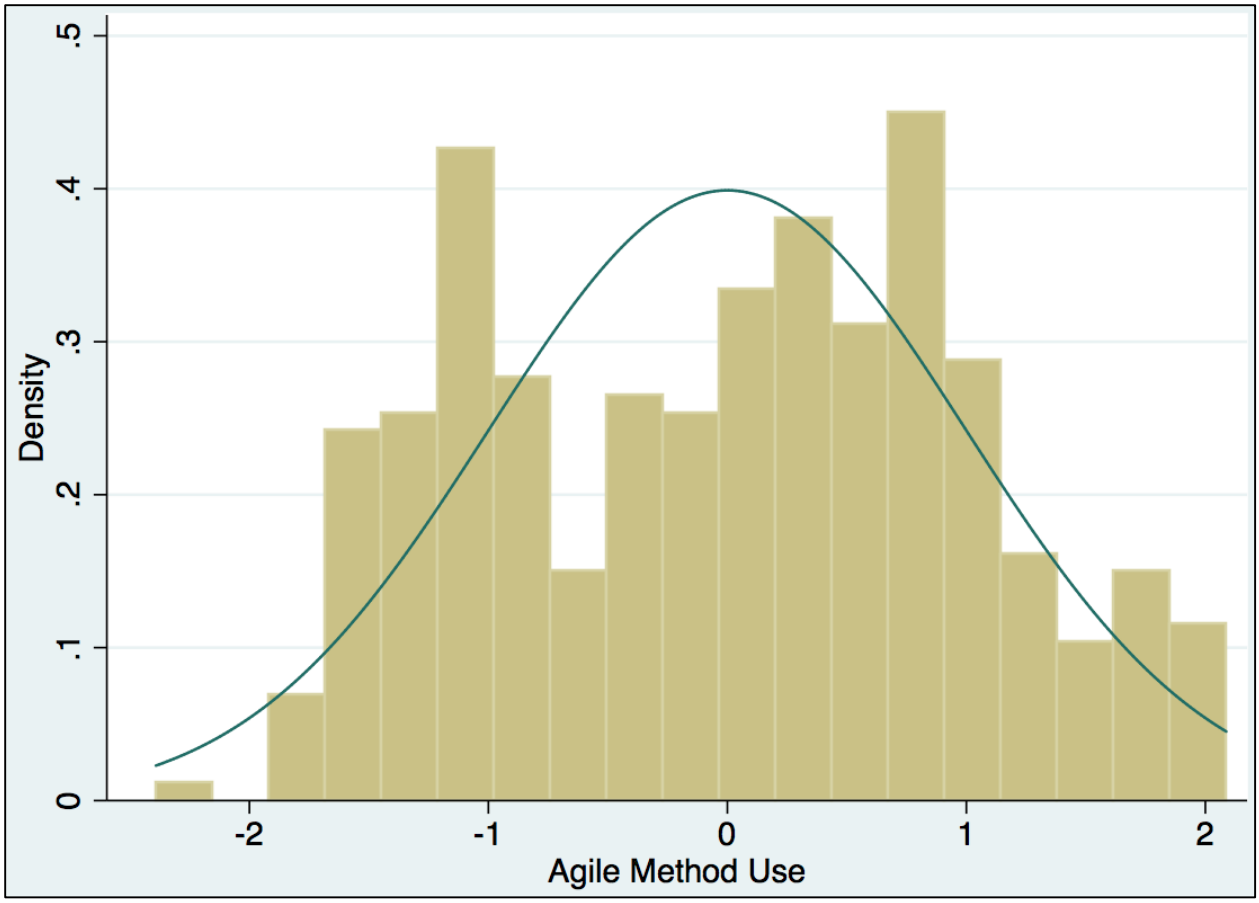

Figure 5.2a: Diagnostic Outputs for Agile Method Use - Histogram

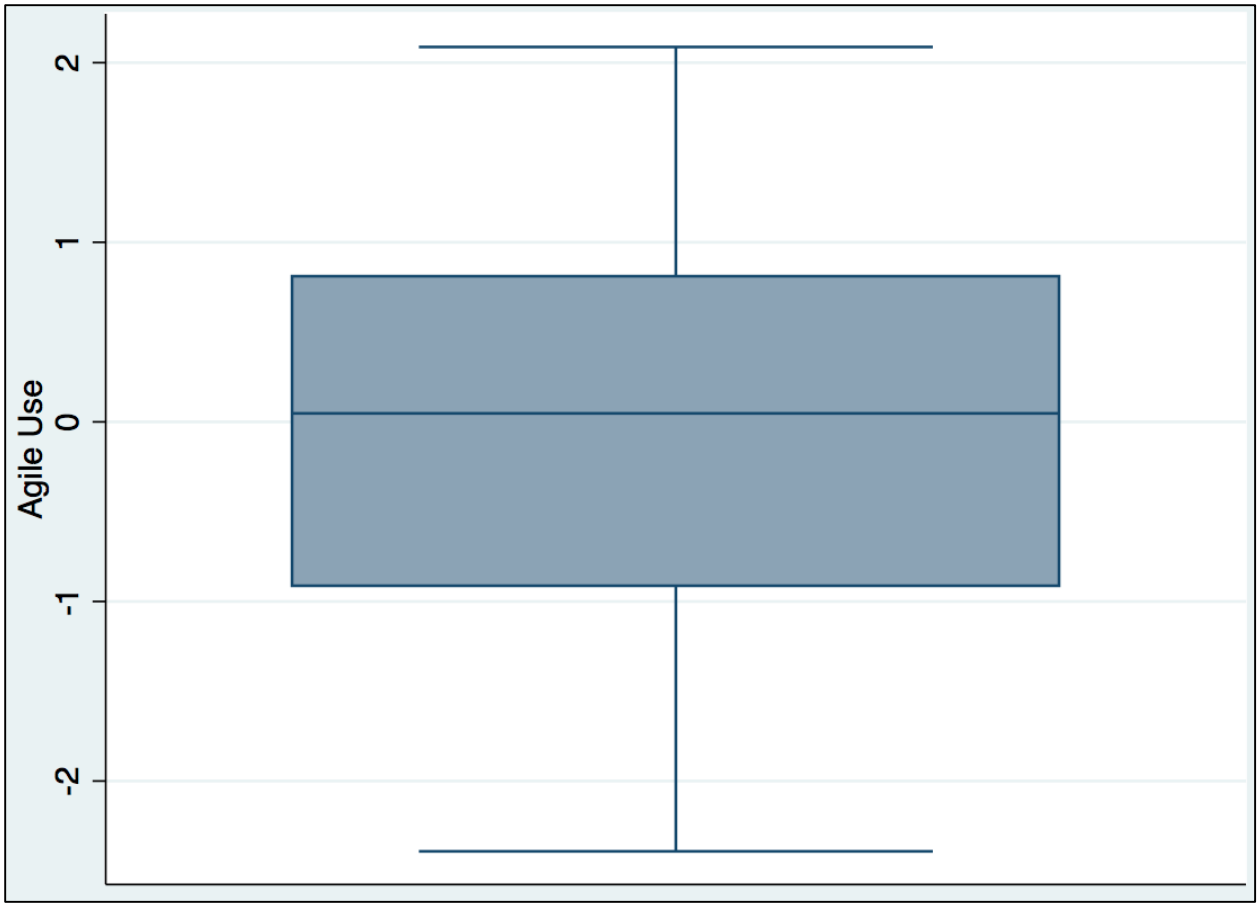

Figure 5.2b: Diagnostic Outputs for Agile Method Use – Box Plot

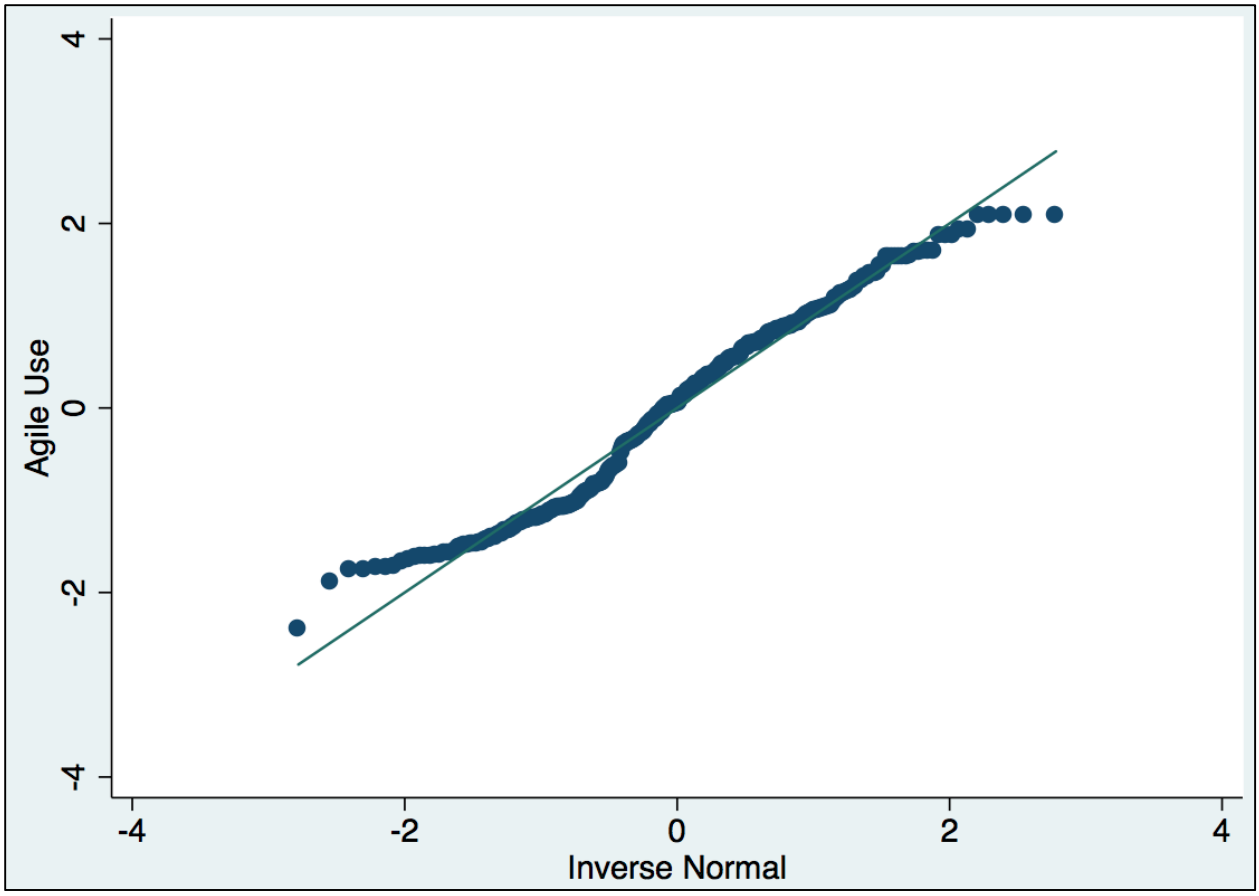

Figure 5.2c: Diagnostic Outputs for Agile Method Use – Q-Normal Plot

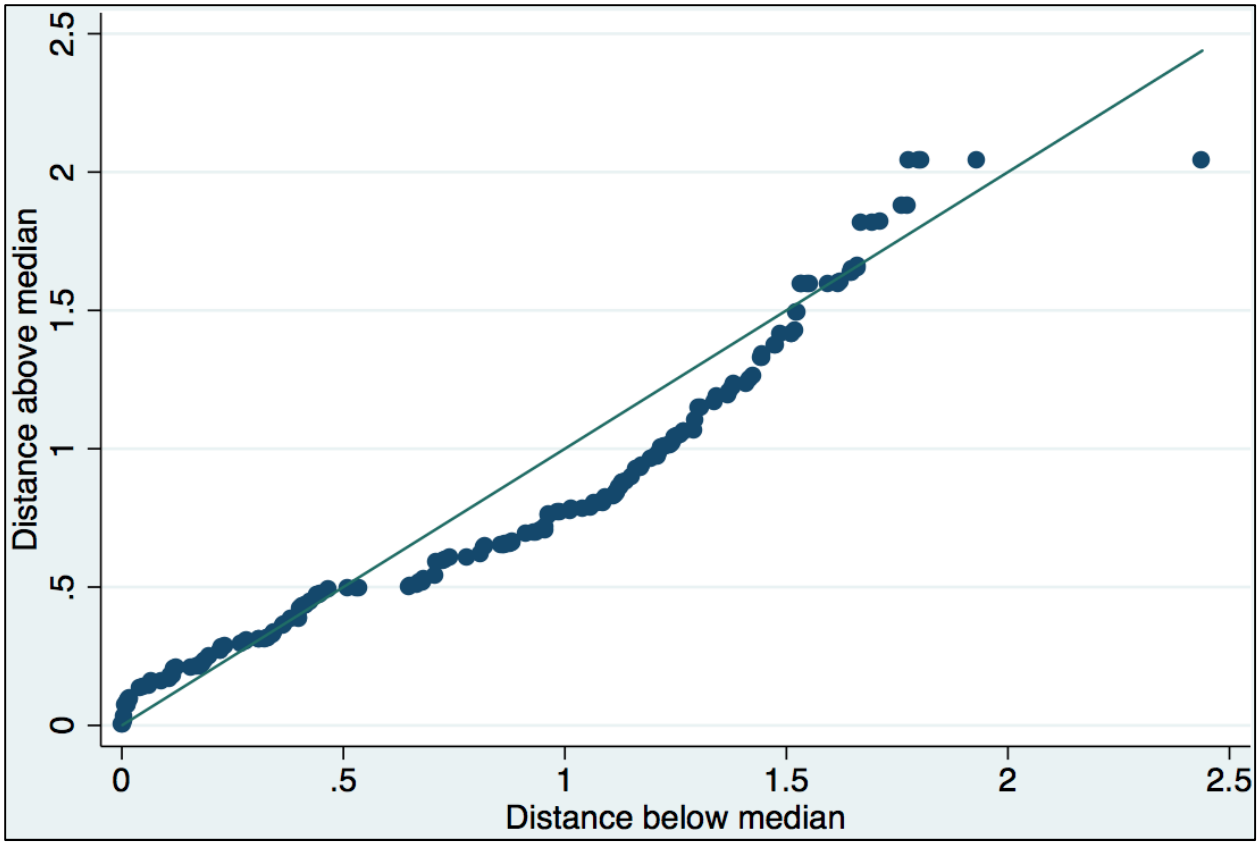

Figure 5.2d: Diagnostic Outputs for Agile Method Use – Symmetry Plot

Agile Method use is also not normal. Based upon the plots, it can be seen that both tails are heavy, and the distribution may have multiple peaks. One outlier may be indicated. Tests for outliers will be discussed below.

Figures 5.3a-d display the diagnostic output for the Dynamism factor. This output indicates the presence of outliers, but besides the outliers, the rest of the plots indicate that Dynamism approximates normality.

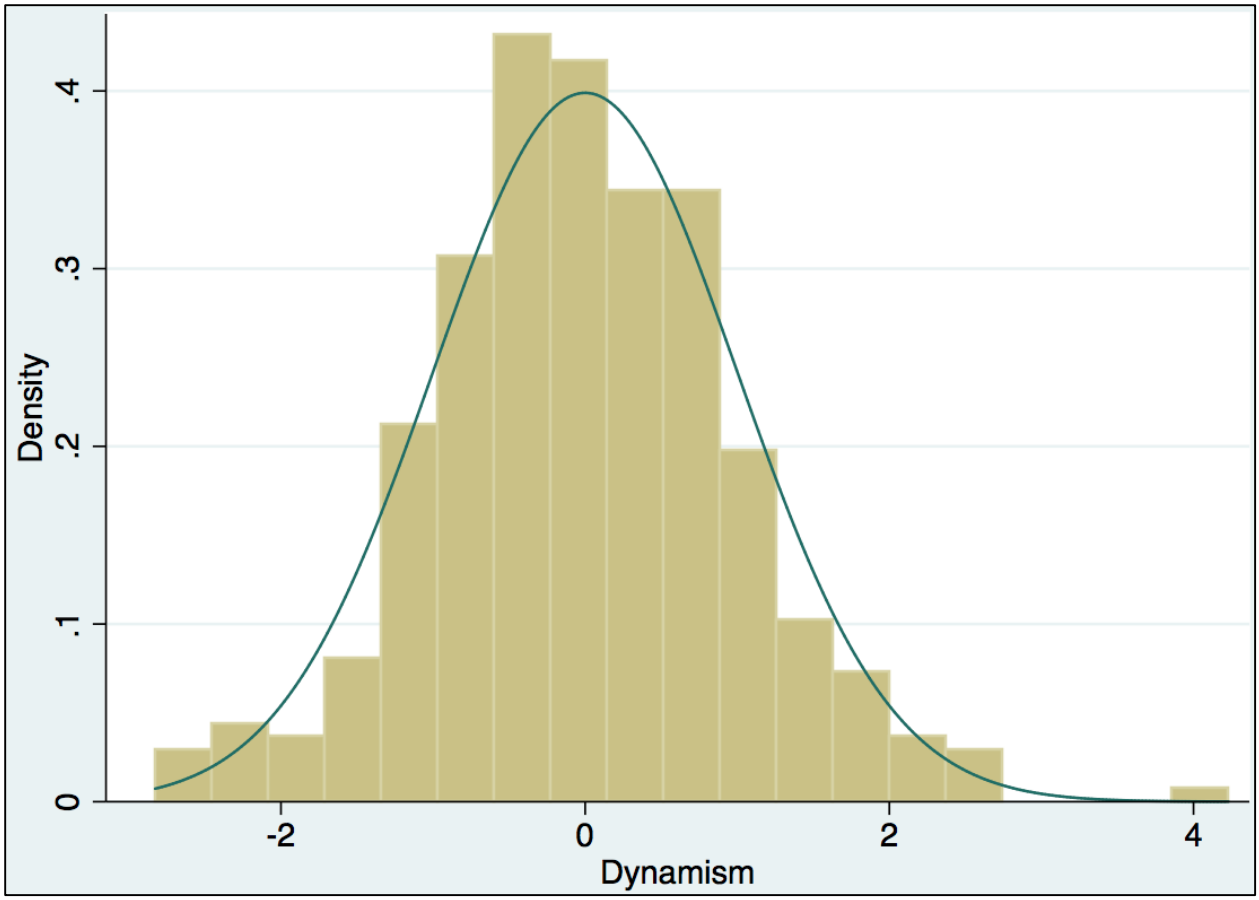

Figure 5.3a: Diagnostic Outputs for Dynamism - Histogram

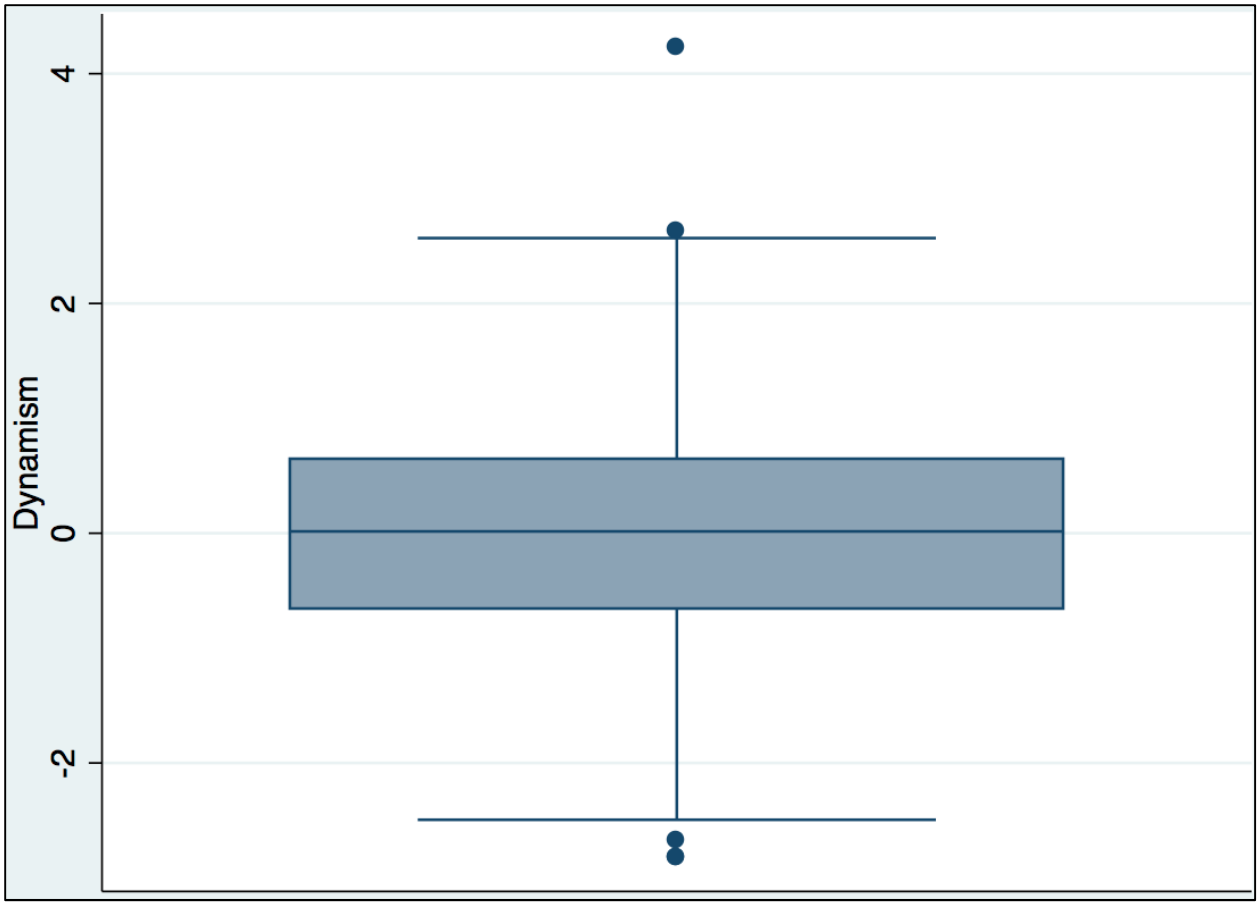

Figure 5.3b: Diagnostic Outputs for Dynamism – Box Plot

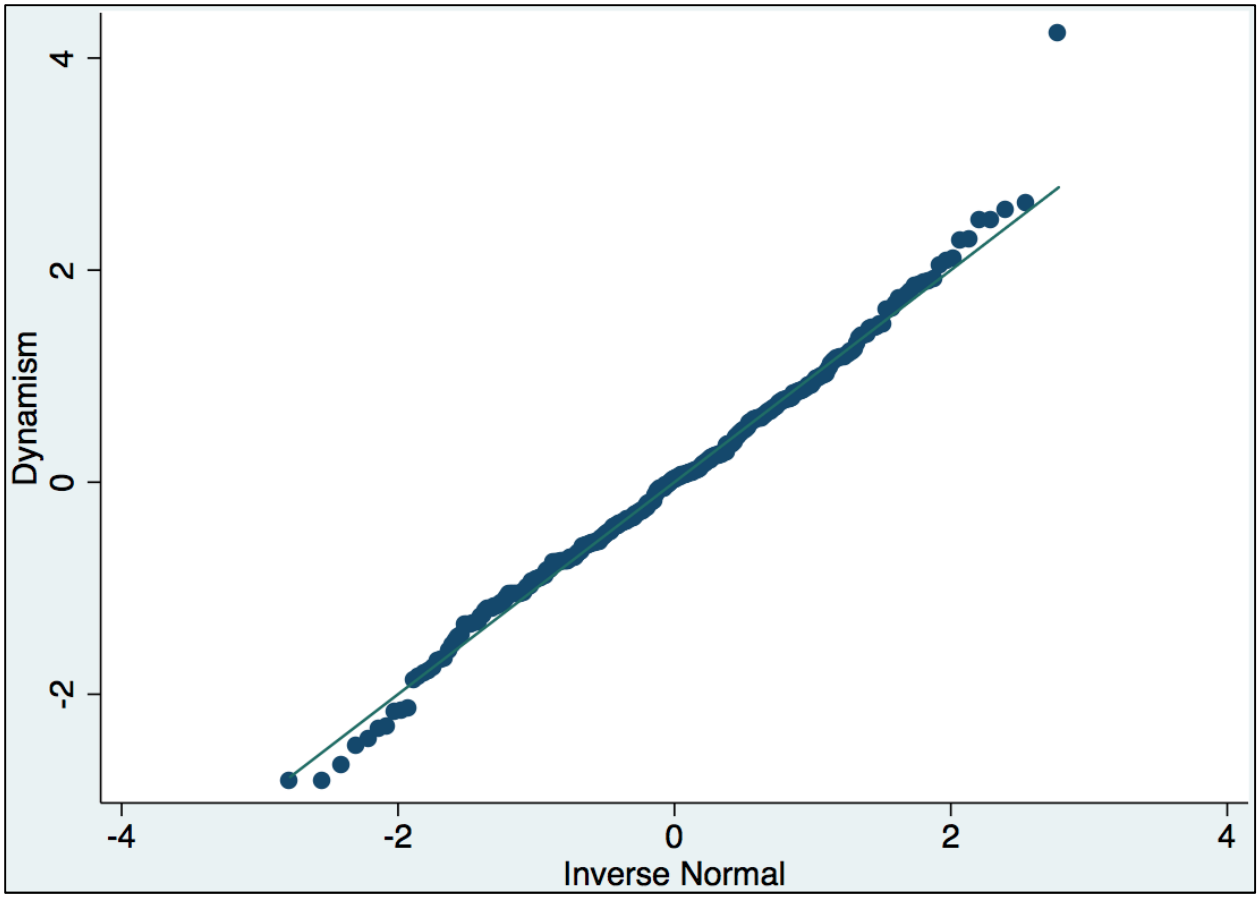

Figure 5.3c: Diagnostic Outputs for Dynamism – Q-Normal Plot

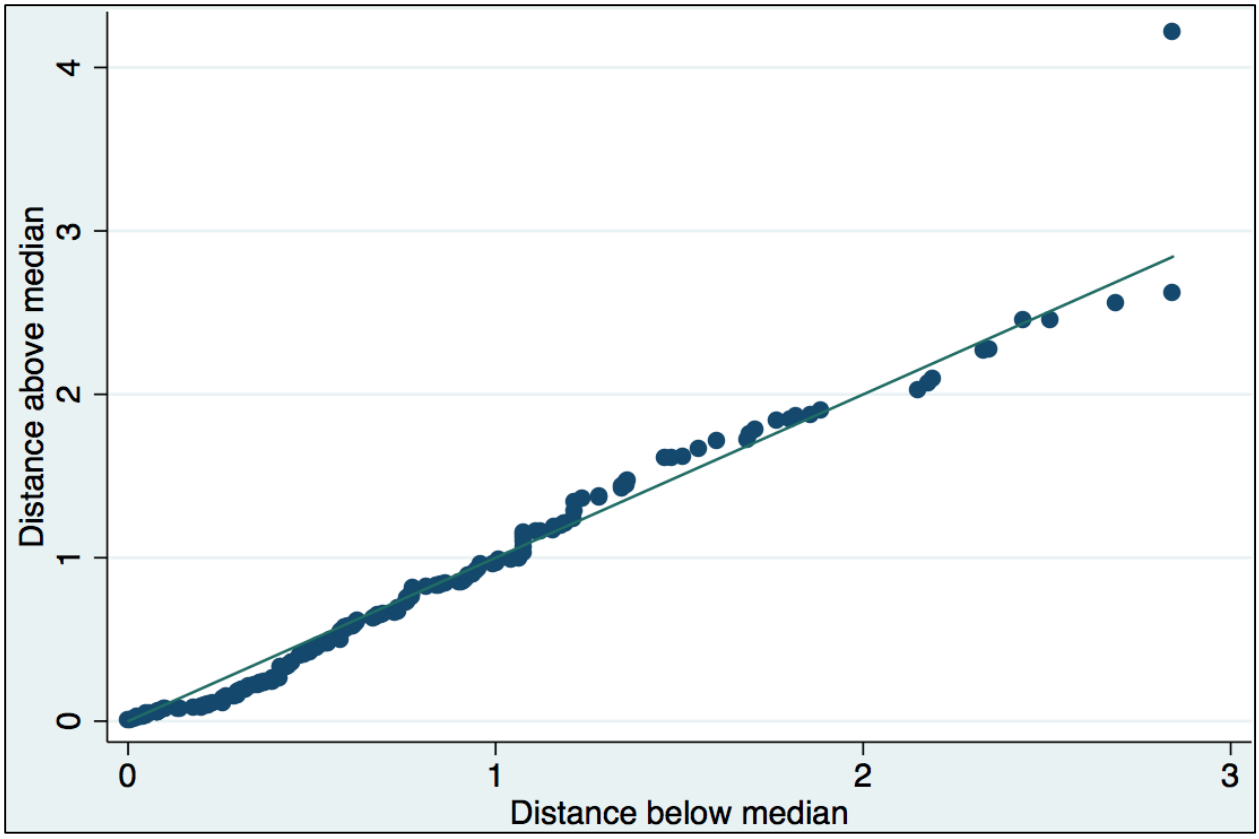

Figure 5.3d: Diagnostic Outputs for Dynamism – Symmetry Plot

Diagnostics for the structural and technical complexity were also performed, which indicated that these distributions were approximately normal, as was expected from the results of the Shapiro-Wilk test.

Tests for Unusual and Influential Data

Next tests for the influence of outliers in the dynamism, agile, and quality data were performed. Figure 5.4 presents a matrix scatter plot for the relationships between each pair of variables. There are several potential outliers indicated in the agile, dynamism, and quality variables. We calculated cooks d for each of the relationships and dropped those cases that exceeded the cutoff for any of the relationships as set by Bollen and Jackman (1990).

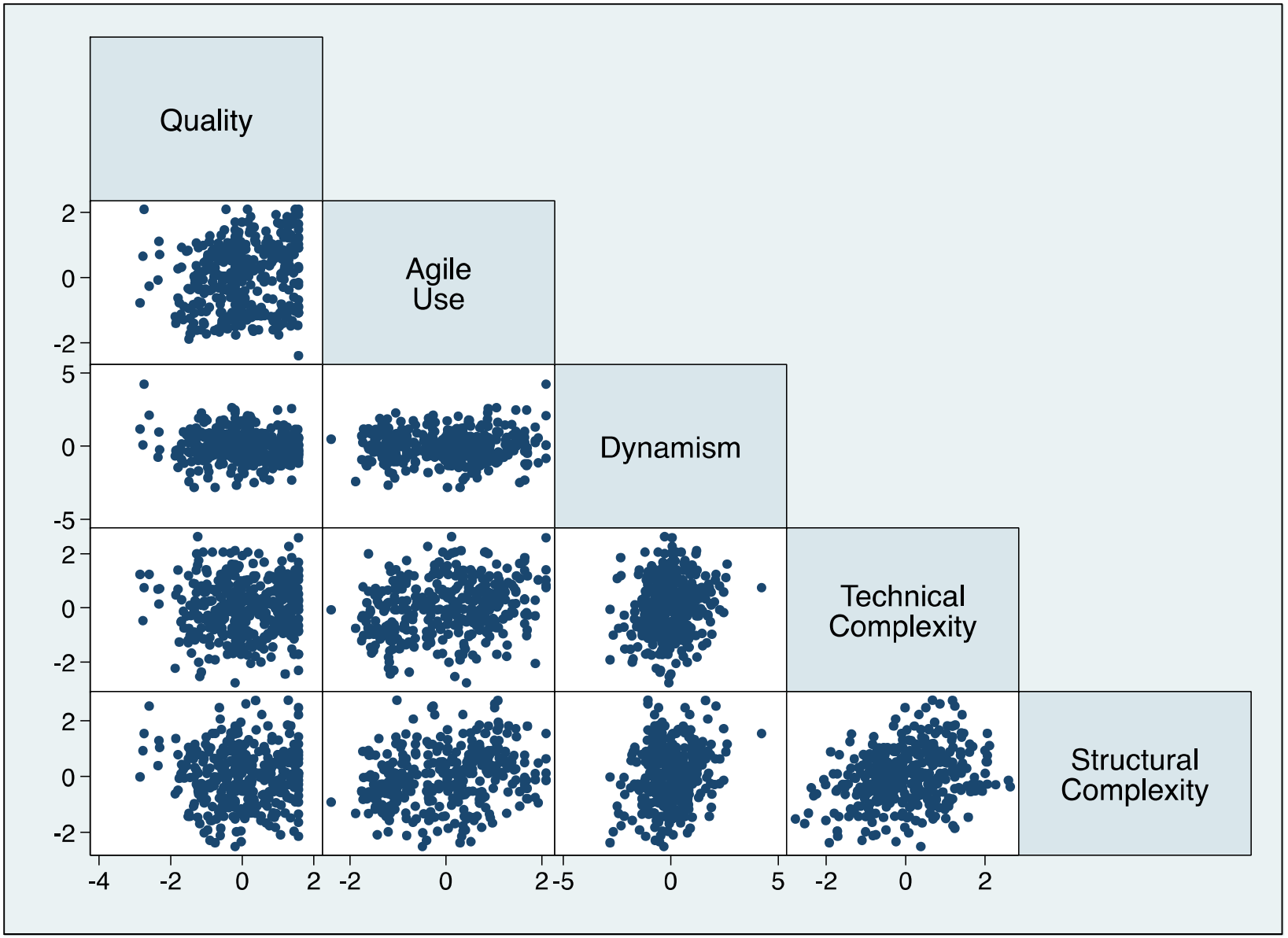

Figure 5.4. Matrix Plot of Factor Relationships

Based upon this analysis, 42 highly influential cases were dropped, leaving the sample with 326 observations. Another Shapiro-Wilk test was performed. No change in results was observed, indicating that influential outliers did not drive the non-normality of the data.

Tests for Linearity

When performing the correlation analysis above, the lack of significant relationships between the factors was noted. One assumption of correlation analysis is the linearity of the relationships between the variables. We performed several analyses of the residuals when performing univariate regressions between the Quality dependent variable and the other factors in the model. Analysis of the homscedascity of residuals and review of augmented component-plus-residual plots indicated that the assumption of linear relationships was unlikely to hold between the predictors and the dependent variables.

In order to investigate the nonlinearity of the relationships between the variables, PROC TRANSREG in SAS was used. PROC TRANSREG allows for regression analysis to be performed with transformed variables. In this analysis, the spline function was chosen, as TRANSREG will fit a spline function transformation to the observed data. Usually, use of a spline function requires the designation of “knots”, or breakpoints in the data that become start and end points for independent estimation of linear relationships for that segment. Rather than manually setting knots for the analysis, PROC TRANSREG fits a transformation to each variable based upon the shape of the observed data. Figure 5.5 shows the transformation plots for the variables in the model.

As can be seen from the plots, the relationships between the variables are nonlinear, and are both bimodal and monotonic.

Due to the fact that the dependent variable is non-normally distributed, and the relationships between the ratio variables are nonlinear, transformation of the data before analysis was necessary.

Transformation of Data

When dealing with non-normal data distributions, a common practice is to attempt to transform the data to create a normal distribution. Several standard transformations were attempted on the Quality, BenFac, and Agile factors, none of which allowed the data to approximate normality.

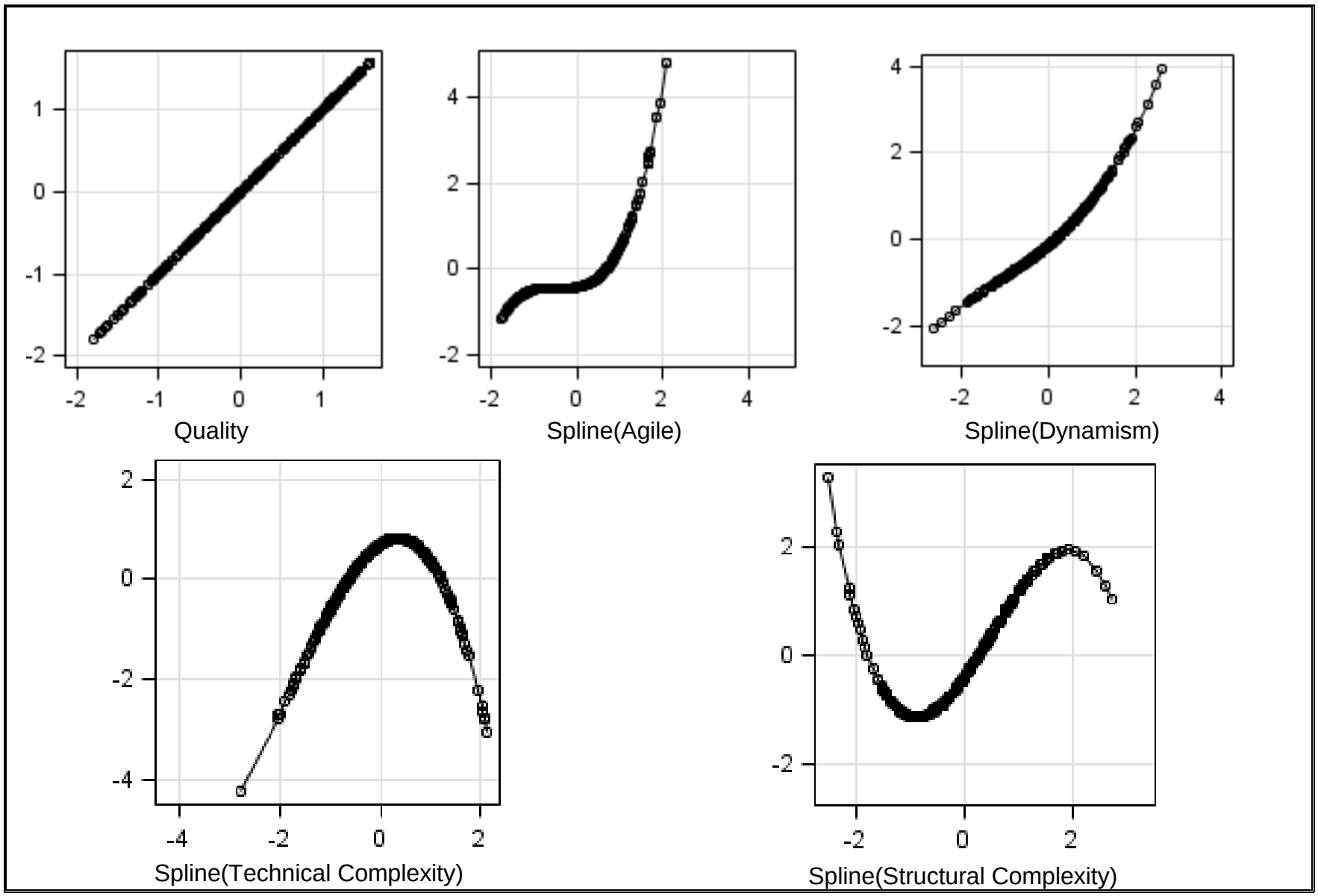

Figure 5.5. Spline Transformation Plots

As the survey items presented the respondents with several likert-scale response options, the original survey data was coded as ordinal. After the factoring and scoring, the variables had been converted to a continuous measure. Because the distance between the intervals in the ordinal categories is arbitrary between subjects, and because the data was originally interpreted by the subjects as ordinal, the factor score data was recoded as ordinal. Ordinal data categorized into at least three categories allows for only slightly less fidelity to generate substantive inferences than does ratio data (Weiss 1986). Because the scale on which the items were answered was a five category scale the data was discretized utilizing five equal width categories. After recoding, all of the discretized variables were again tested with the Shapiro-Wilk test, and the null hypothesis that the variables were normal could not be rejected for any of them.

Analysis Results

This section presents the statistical results for the data analysis that was performed to test the hypotheses of the theoretical model.

Because the theory and data are multilevel in nature (i.e. – people in project teams), the sample violates several statistical assumptions. The first is that the observations are independent, and that the random errors are independent, homoscedastic and normally distributed. If the sample was tested without regard to the expected similarity of responses it would imply that the members of a group share no common attitudes or other characteristics that might influence their responses regarding the project. This is specifically contrary to the assumption about the responses from the same group. Alternatively, while the data could be aggregated a single group-level

mean be utilized for the analysis, this would reduce statistical power, and potentially provide biased estimates of group-level relationships, and lead to incorrect causal inferences (Bovaird 2007).

There are several robust techniques to test multilevel models, and to separate the between group variance (random effects) from the within group variance (fixed effects). However, these methodologies require sufficient sample size at each level. We attempted to perform multilevel modeling with four different statistical packages and seven different analysis techniques. While first-level effects were successfully generated in all of the tools, none could fit the second-level effects.

Consequently, a population-averaged model was utilized. This type of analysis fits a single model to all of the data, but controls for within-cluster correlation. While this type of model does not estimate the within and between team variance like a multilevel model would, it uses the concept of nesting to correct for the presence of similarity within clusters of responses. The data in the sample is made up of people within project teams. Because of this, it would be expected that their responses would be similar. In order to correct for this nested effect, we used a nested generalized linear model to test the second-order discretized factor data. We discretized the data due to the severe non-normality of several of the predictors and the nonlinear relationships between the variables. Additionally, the sample size limitations, limited testing each hypothesis with an independent model, rather than testing the three moderating hypotheses simultaneously.

After dropping the influential cases described above, the sample reduced to 326 cases. After this, any project team for which only one case remained was also dropped.

The final analysis was performed using 305 responses, and controlled for nesting within 56 teams.

Testing the Research Model

We proposed that the agile methods would positively impact project success, and that the dimensions of uncertainty would moderate the impact of agile method use on project success. These hypotheses were as follows:

H1: The extent of agile methodology use will positively impact project success

H2a: Structural complexity will negatively moderate the impact of agile methodology use on project success

H2b: Technical complexity will negatively moderate the impact of agile methodology use on project success

H2c: Environmental dynamism will positively moderate the impact of agile methodology use on project success

In order to explore the direct impacts of the extent of Agile Method Use on Project Success, and to explore the moderating effects of the dimensions of uncertainty, three models were tested. As described above, the limited sample size at the team level did not allow us to test all the moderation effects simultaneously. Each of the direct and moderation effects was tested in an independent model, as indicated in Table 5.12.

Structural Complexity

The results of Model 1 are presented below. First, Table 5.13 provides goodness of fit statistics for each of the dependent variables, with the extent of agile method use as the independent variable, and structural complexity as the moderator. All models have a non-significant chi-square test, indicating reasonable goodness of fit, and the

omnibus test indicates that the model parameters are not all zero. Table 5.13 also presents the tests of model effects (Wald Chi-Square tests) by parameter, both at the individual and nested levels. This test describes the effect of the variable as a whole. As can be seen, the explanatory variables are significant in all five models.

Table 5.12. Generalized Linear Models Used to Test Moderating Effects of Uncertainty

Variable	Model 1a-e	Model 2a-e	Model 3a-e
Quality	A	A	A
Benefits	B	B	B
Budget Outcome	C	C	C
Time Outcome	D	D	D
Scope Outcome	E	E	E
Control Variables	Yes	Yes	Yes
Extent of Agile Method Use	Yes	Yes	Yes
Structural Complexity	Yes		
Agile * Structural Complexity	Yes		
Technical Complexity		Yes	
Agile * Technical Complexity		Yes	
Dynamism			Yes
Agile * Dynamism			Yes
Nested Effects	Yes	Yes	Yes

Table 5.14 presents the tests of model effects by parameter level, as compared with the reference level of each variable (=5). The parameter estimates indicate that the relationship between the extent of agile method use and structural complexity and the outcome variables is significant at almost all levels. Further the parameters indicate that the relationships between the variables are likely to be nonlinear, as the trends across levels are not of a consistent direction and magnitude. Even with this nonlinear structure, and even while some levels are insignificant, Table 5.19 indicates that each variable is significant as a whole.

Table 5.13. Goodness of Fit of Model 3 the Moderation Effects of Structural Complexity.

Model Fit	Quality	Benefits	Budget Outcome	Time Outcome	Scope Outcome
Model Chi-Square	6.198	5.087	7.804	18.670	7.241
(df, sig.)	(10, .620)	(10, .509)	(10, .780)	(10, 1.867)	(10, .724)
Model Omnibus Test	1096.959	1211.773	1310.769	958.286	1202.672
(df, sig.)	(294, .000)	(294, .000)	(294, .000)	(294, .000)	(294, .000)
Population Effects (Wald Chi-Square Test)					
Intercept	24.613**	14.040**	10.505**	65.916**	159.387**
Agile	320.720**	168.532**	217.513**	58.828**	198.608**
Structural Complexity	46.303**	247.533**	133.946**	66.576**	295.777**
Agile * SC	73.993**	240.409**	112.217**	35.457**	103.381**
Management Support	54.791**	170.755**	94.613**	32.740**	197.756**
Project Length	35.993**	9.887**	31.349**	1.593	13.408**
Organization Size	61.706**	104.574**	50.678**	.025	41.892**
Org. Agile Experience	35.362**	40.211**	40.322**	10.268**	114.721**
Respondent Experience	66.037**	.903	28.178**	44.494**	24.271**
Respondent Agile Exp.	23.668**	44.618**	71.738**	15.699**	31.295**
Nested Effects (Wald Chi-Square Test)					
Agile	475.706**	586.369**	854.574**	211.290**	754.558**
Structural Complexity	793.760**	915.211**	1470.651**	310.789**	617.564**
Agile * SC	125.267**	81.211**	41.658**	49.599**	107.967**
Management Support	344.725**	211.007**	623.352**	204.445**	633.424**
Respondent Experience	53.717**	175.248**	46.716**	42.157**	156.380**
Respondent Agile Exp.	126.562**	266.256**	252.030**	61.091**	292.425**

Table 5.14. Parameter Estimates for Model 3, the Moderation Effects of Structural Complexity.

	Quality	Benefits	Budget Outcome	Time Outcome	Scope Outcome
Intercept	5.976**	1.0896**	-1.886	-4.875*	-13.491**
Agile=1	-21.820**	4.1921**	19.447**	13.340	44.661**
Agile=2	-8.916**	1.2445**	5.140**	10.734**	17.381**
Agile=3	-19.636**	1.9745**	1.216	21.870**	15.536**
Agile=4	-3.531**	.7041**	-.436	4.096**	7.058**
Agile=5	0	0	0	0	0
Structural Complexity=1	5.716**	1.7592**	3.237	3.759	14.783**
Structural Complexity=2	8.494	5.2091**	-21.760**	-4.912	-41.290**
Structural Complexity=3	15.649**	3.8085**	-18.194**	-9.048	-35.551**
Structural Complexity=4	18.266**	3.9590**	-18.400**	-10.347	-40.676**
Structural Complexity=5	0	0	0	0	0
Management Support=1	-.474	-2.111**	1.858**	-1.047	1.566**
Management Support=2	-1.077*	2.721**	-.880	-1.113	2.111**
Management Support=3	-.991**	-2.862**	-.761**	-1.463**	2.075**
Management Support=4	-.813*	-6.353**	-.893*	-3.126**	.059
Management Support=5	0	0	0	0	0
Project Length	-.172**	.082**	.180**	.063	.114**
Organization Size	.161**	.189**	.163**	-.006	-.143**
Org. Agile Experience	.508**	-.491**	-.609**	-.475**	-.990**
Respondent Experience	.012	.629**	.136**	.264**	.512**
Respondent Agile Exp.	-1.302**	-1.198**	.712**	.933**	1.639**

Next the moderation effect of structural complexity on the impact of the extent of agile method use on project success was evaluated. Because the moderation is tested using discretized data, the moderation effect is reported by combination of levels from each variable. The breakdown of cases by category is presented in Table 5.15.

Table 5.15. Cases by Category – Structural Complexity

	Agile					
	=1	=2	=3	=4	=5	Total
Structural Complexity=1	0	1	2	0	1	4
Structural Complexity=2	4	14	12	6	1	37
Structural Complexity=3	13	59	63	72	10	217
Structural Complexity=4	0	8	13	11	8	40
Structural Complexity=5	0	0	4	3	0	7
Total	17	82	94	92	20	305

Remember that the hypothesis for the moderating impacts of structural complexity was in the negative direction, but the negative impacts of project management outcomes indicate increases from baseline estimates. This means that, for the project management outcome variables, a positive beta coefficient indicates a negative (as hypothesized) moderation effect. For this reason, the summary of interaction terms is presented using the terms of hypothesized vs. reversed direction of effect, rather than as positive or negative. Table 5.16 presents a summary of the directions of interaction effects.

Table 5.16. Summary of Interaction Effects - Structural Complexity

Model	Reversed Direction	Not Significant	Hypothesized Direction
Quality	4	3	6
Organizational Benefits	2	1	10
Budget Outcome	1	2	10
Time Outcome	3	8	2
Scope Outcome	1	1	11
Total, (%)	11 (17%)	15 (23%)	39 (60%)

Table 5.17 provides support for the negative moderating impact of structural complexity on project success. At each level of Structural Complexity, the anticipated effect is usually in the direction predicted. Interestingly, those impacts that are reversed from the predicted hypothesis all appear at the lowest level of agile method use or the lowest level of structural complexity. As shown in Table 5.16, these groups had the fewest number of cases, as few as 1. Because of this, caution should be used when interpreting differences regarding these groups.

The interaction terms indicate the predicted difference in effect from the reference categories. As can be seen from the table, interactions are significant at all levels for the Quality and Organization Benefits models. Fewer of the interactions are significant for the project management success indicators. Given the observation discussed above of the nonlinear relationships between both the IV and the moderators with the outcome variables, the fact that the interactions are significant but are, at times, in a direction opposite to that hypothesized is not surprising.

Table 5.17. Nested Model Interaction Coefficients – Structural Complexity

Quality			Agile		
	=1	=2	=3	=4	=5
Structural Complexity=1	a	23.615**	12.222**	a	0
Structural Complexity=2	6.743**	-2.741	-12.883*	-27.019**	0
Structural Complexity=3	1.185**	-12.189**	-18.063**	-14.751**	0
Structural Complexity=4	a	3.200	-3.173	-14.749**	0
Structural Complexity=5	0	a	0	0	0

Org Benefits			Agile		
	=1	=2	=3	=4	=5
Structural Complexity=1	a	-6.605**	9.489**	a	0
Structural Complexity=2	22.967**	-24.937**	-12.873*	-35.317**	0
Structural Complexity=3	1.367**	-44.658**	-25.011**	-25.796**	0
Structural Complexity=4	a	-28.441**	-3.946	-24.529**	0
Structural Complexity=5	0	a	0	0	0

Budget Outcome			Agile		
	=1	=2	=3	=4	=5
Structural Complexity=1	a	-15.507**	-.735	a	0
Structural Complexity=2	11.391**	16.330**	30.182**	31.328**	0
Structural Complexity=3	4.982**	11.277*	22.891**	20.745**	0
Structural Complexity=4	a	11.593	21.276**	19.316**	0
Structural Complexity=5	0	a	0	0	0

Time Outcome			Agile		
	=1	=2	=3	=4	=5
Structural Complexity=1	a	-28.845**	-9.535**	a	0
Structural Complexity=2	10.247**	-4.066	11.880	21.852*	0
Structural Complexity=3	4.601**	-1.148	9.892	9.178	0
Structural Complexity=4	a	-11.030	-3.302	8.304	0
Structural Complexity=5	0	a	0	0	0

Scope Outcome			Agile		
	=1	=2	=3	=4	=5
Structural Complexity=1	a	-28.663**	-.084	a	0
Structural Complexity=2	28.004**	27.690**	50.471**	45.697**	0
Structural Complexity=3	6.886**	18.625**	36.084**	34.392**	0
Structural Complexity=4	a	20.828**	39.883**	36.140**	0
Structural Complexity=5	0	a	0	0	0

a: no observations, 0: reference group,* p<.05, ** p<.01

Summary of results: Hypothesis 2a

Structural Complexity will negatively moderate the impacts of Agile Method Use on Project Success.

The moderating effects of structural complexity on agile method use were found to be significant on all components of project success. The relationship between structural complexity and project success is bi-modal. Because of this nonlinearity, it was likely that mixed results would be observed when testing for the moderating effects of structural complexity. Even so, it was found that 60% of the tested combinations of level interactions were significantly positive and, of those combinations that were significant, 78% of them were positive in direction. The presence of such a high proportion of positive effects in the presence of such strong nonlinearity lends strong support for Hypothesis 2a.

Technical Complexity

Model 2 was utilized to investigate the negative moderation effects of technical complexity on the extent of agile method use. The method and interpretation used for Model 2 and Model 3 was identical to that for Model 1.

Table 5.18 provides goodness of fit statistics and tests of model effects for each of the Model 4 instances. As in the case of Model 1, all instances of Model 2 have a non-significant chi-square test, indicating reasonable goodness of fit, and the omnibus test indicates that the model parameters are not all zero. The tests of model fit indicate that the explanatory variables still maintain significance for all of the model instances, although control variables are not significant on the project management success models.

Table 5.19 presents the parameter estimates and significance tests for the direct effects of agile methodologies and technical complexity.

Table 5.18. Goodness of Fit of Model 4. Moderation Effects of Technical Complexity.

Model Fit	Quality	Benefits	Budget Outcome	Time Outcome	Scope Outcome
Model Chi-Square	14.287	14.446	35.698	36.551	15.333
(df, sig.)	(19, .757)	(19.760)	(19, 1.879)	(19, 1.924)	(19, .807)
Model Omnibus Test	842.249	893.450	847.032	753.388	973.834
(df, sig.)	(285, .000)	(285, .000)	(285, .000)	(285, .000)	(285, .000)
Population Effects (Wald Chi-Square Test)					
Intercept	12.078**	.017	19.560**	74.054**	129.852**
Agile	76.579**	10.828*	21.152**	19.595**	20.304**
Technical Complexity	60.519**	87.206**	35.210**	13.989**	20.595**
Agile * TC	19.328**	46.226**	7.501	8.698*	14.422**
Management Support	57.910**	69.760**	30.019**	15.877**	53.707**
Project Length	24.128**	4.010*	13.536**	5.358	23.809**
Organization Size	25.241**	39.154**	.749	10.170*	.781
Org. Agile Experience	48.455**	15.321**	.053	28.891**	6.831**
Respondent Experience	20.898**	12.615**	.406	23.053**	.135
Respondent Agile Exp.	4.142*	37.548**	1.046	1.798	.365
Nested Effects (Wald Chi-Square Test)					
Agile	544.492**	374.396**	221.027**	189.664**	507.173**
Technical Complexity	215.142**	182.395**	103.238**	105.983**	235.176**
Agile * TC	66.118**				
Management Support	302.494**	371.655**	247.582**	141.362**	298.996**
Respondent Experience	252.087**	112.135**	45.023**	31.082**	140.598**
Respondent Agile Exp.	168.169**	145.091**	67.655**	35.391**	65.502**

Table 5.19. Parameter Estimates for Model 4, the Moderation Effects of Technical Complexity.

	Quality	Benefits	Budget Outcome	Time Outcome	Scope Outcome
Intercept	6.093**	-.862	3.382	7.906**	1.071
Agile=1	-11.202**	-5.857**	1.582	-2.487	-.110
Agile=2	-10.388**	-12.245**	2.379	-10.201**	-3.007
Agile=3	-4.462*	4.638*	5.014	1.194	5.558**
Agile=4	-10.149**	-10.392**	5.237	-4.540	-3.306
Agile=5	0	0	0	0	0
Technical Complexity=1	-4.767*	5.845**	1.026	-.106	5.605**
Technical Complexity=2	-.024	.847	-.931	.997	.867
Technical Complexity=3	4.001**	6.967**	-3.458	.734	.209
Technical Complexity=4	1.144**	.371	-1.575	-1.295**	-.883**
Technical Complexity=5	0	0	0	0	0
Management Support=1	-.020	2.232**	-2.343	-.010	1.357**
Management Support=2	-.366	.293	-2.560	-2.004*	.070
Management Support=3	.044	.205	-1.781	-.657	.505**
Management Support=4	-.464	-3.308**	-1.255	-3.125**	-1.701
Management Support=5	0	0	0[a]	0	0
Project Length	.171**	-.070*	.202	.129*	-.176**
Organization Size	.148**	.185**	.040	.150**	.027
Org. Agile Experience	-.836**	-.473**	-.044	-1.032**	-.325**
Respondent Experience	.249**	.614**	.016	.406**	.436**
Respondent Agile Exp.	-.298**	-.055	-.074	-.235**	.005

Table 5.20 lists the breakdown of cases by category for technical complexity. In comparing this breakdown to Table 5.15, the cases are more evenly dispersed, although there are still three cells with no cases.

Table 5.20. Cases by Category – Technical Complexity

	Agile					
	=1	=2	=3	=4	=5	Total
Technical Complexity=1	0	3	9	4	0	16
Technical Complexity=2	7	27	27	14	5	80
Technical Complexity=3	6	38	36	23	12	115
Technical Complexity=4	4	13	18	38	2	75
Technical Complexity=5	0	1	4	13	1	19
Total	17	82	94	92	20	305

Table 5.21 lists the summary of directionality of interaction effects for Technical Complexity. Only 16% of the interactions at the discrete levels are in the hypothesized direction, while 70% of the interactions are not significant.

Table 5.21. Summary of Interaction Effects - Technical Complexity

Model	Reversed Direction	Not Significant	Hypothesized Direction
Quality	2	6	2
Organizational Benefits	1	3	6
Budget Outcome	1	9	0
Time Outcome	3	7	0
Scope Outcome	0	10	0
Total, (%)	7 (14%)	35 (70%)	8 (16%)

Table 5.22 presents the specific interaction results by model and by discrete category. The results presented in Table 5.22 provide little support for the hypothesis of a negative moderation effect of technical complexity.

Table 5.22. Nested Model Interaction Coefficients – Technical Complexity

Quality			Agile		
	=1	=2	=3	=4	=5
Technical Complexity =1	a	9.655**	2.262	0	a
Technical Complexity =2	5.211**	1.829	-2.931	-1.298	0
Technical Complexity =3	0	-1.452	-4.624**	-.811	0
Technical Complexity =4	0	0	-2.920**	0	0
Technical Complexity =5	a	0	0	0	0
Org Benefits			Agile		
	=1	=2	=3	=4	=5
Technical Complexity =1	a	.738	-17.143**	0	a
Technical Complexity =2	5.444**	.522	-8.073**	-2.738*	0
Technical Complexity =3	0	-1.564	-12.167**	-3.277**	0
Technical Complexity =4	0	0	-8.007**	0	0
Technical Complexity =5	a	0	0	0	0
Budget Outcome			Agile		
	=1	=2	=3	=4	=5
Technical Complexity =1	a	-.827	-9.532*	0	a
Technical Complexity =2	3.061	-.760	-2.483	-.490	0
Technical Complexity =3	0	.147	-.797	-1.046	0
Technical Complexity =4	0	0	-.263	0	0
Technical Complexity =5	a	0	0	0	0
Time Outcome			Agile		
	=1	=2	=3	=4	=5
Technical Complexity =1	a	5.819	-5.262	0	a
Technical Complexity =2	-1.984	-1.990	-7.656**	-6.787**	0
Technical Complexity =3	0	1.390	-4.859**	-3.211	0
Technical Complexity =4	0	0	-2.448	0	0
Technical Complexity =5	a	0	0	0	0
Scope Outcome			Agile		
	=1	=2	=3	=4	=5
Technical Complexity =1	a	-4.102	-10.602	0	a
Technical Complexity =2	.516	-5.010	-4.687	-3.654	0
Technical Complexity =3	0	-1.503	-2.910	.038	0
Technical Complexity =4	0	0	-2.723	0	0
Technical Complexity =5	a	0	0	0	0

a: no observations, 0: reference group,* p<.05, ** p<.01

Summary of results: Hypothesis 2b

Technical Complexity will negatively moderate the impact of Agile Method Use on Project Success

We find partial support for the relationship between technical complexity and project success. This relationship is an inverse u shaped curve. This means that when technical complexity is very low or is very high, the relationship with project success is negative, but as the level of technical complexity approaches the mean, the relationship becomes positive. Because of this nonlinearity, mixed results of hypothesis testing were observed.

Of the tested combinations, only 16% of the tests were significant and in the hypothesized direction. Of the remaining combinations, 70% were found to be insignificant, and 14% were significant in the opposite direction as was predicted.

These results indicate that support for hypothesis is weak for most dimensions of project success, with the exception of organizational benefits, for which 86% of the significant interaction combinations were positive.

Dynamism

Model 3 was used to complete the tests of the moderation effects of uncertainty on project success. In contrast to hypotheses 2a and 2b, the moderation effects of dynamism were predicted to be positive. Table 5.29 provides goodness of fit statistics for each of the dependent variables, with the extent of as the independent variable, and dynamism as the moderator. Again, all models have a non-significant chi-square test, indicating reasonable goodness of fit, and the omnibus test indicates that the model parameters are not all zero.

Table 5.23 presents the tests of model effects by parameter, both at the individual and nested levels. As before, this test evaluates the effect of the variable as a whole. Table 5.24 lists the breakdown of cases by category for technical complexity.

The hypothesis for the moderating impacts of dynamism was in the positive direction, but the positive impacts of project management outcomes indicate decreases from baseline estimates. This means that, for the project management outcome variables, a negative beta coefficient indicates a positive (as hypothesized) moderation effect. Table 5.32 lists the summary of directionality of interaction effects for Technical Complexity. About 46% of the interactions at the discrete levels are in the hypothesized direction, while 14% of the interactions in the opposite direction from the hypothesis. Of those combinations that had significant effects, 70% were in the hypothesized direction. As the spline transformation regression showed a nearly linear positive relationship between dynamism and quality, these results are not unexpected.

Table 5.23. Goodness of Fit of Model 5, the Moderation Effects of Dynamism.

Model Fit	Quality	Benefits	Budget Outcome	Time Outcome	Scope Outcome
Model Chi-Square	5.635	1.809	11.550	9.521	4.474
(df, sig.)	(10, .563)	(10, .181)	(10, 1.155)	(10, .952)	(10, .447)
Model Omnibus Test	1126.022	1527.079	1191.213	1163.668	1349.540
(df, sig.)	(294, .000)	(294, .000)	(294, .000)	(294, .000)	(294, .000)
Population Effects (Wald Chi-Square Test)					
Intercept	61.566**	96.116**	2.013	2.641	16.498**
Agile	263.890**	909.798**	37.195**	133.621**	224.609**
Dynamism	93.424**	906.354**	102.049**	45.870**	87.185**
Agile * Dynamism	225.243**	877.381**	198.021**	163.838**	390.676**
Management Support	231.467**	533.074**	65.890**	54.667**	61.484**
Project Length	69.305**	92.106**	48.395**	29.170**	.469
Organization Size	11.180**	141.432**	1.813	52.786**	.562
Org. Agile Experience	27.006**	28.762**	1.116	24.686**	15.859**
Respondent Experience	74.014**	117.899**	5.520*	.054	31.824**
Respondent Agile Exp.	46.127**	89.873**	1.248	2.017	33.239**
Nested Effects (Wald Chi-Square Test)					
Agile	310.885**	1007.804**	393.487**	521.989**	378.755**
Technical Complexity	1078.846**	1999.906**	847.630**	711.497**	881.098**
Agile * Dynamism					
Management Support	435.109**	1525.429**	269.694**	134.000**	169.585**
Respondent Experience	313.788**	765.330**	81.166**	64.763**	56.396**
Respondent Agile Exp.	146.601**	803.866**	206.406**	200.650**	54.435**

Table 5.24. Parameter Estimates for Model 5, the Moderation Effects of Dynamism.

	Quality	Benefits	Budget Outcome	Time Outcome	Scope Outcome
Intercept	-6.164**	-6.966**	-1.543	1.692*	4.065**
Agile=1	-54.951**	-.005	-42.326**	50.735**	-12.813
Agile=2	-27.354**	16.122**	-3.138	44.376**	19.658**
Agile=3	-32.324**	6.084	-6.905	33.635**	13.831*
Agile=4	14.718**	20.000**	5.631**	-1.135**	7.201**
Agile=5	0	0	0	0	0
Dynamism=1	-36.051**	10.187*	-19.233	50.690**	6.399
Dynamism=2	-37.727**	9.730*	-18.817	52.111**	6.886
Dynamism=3	-103.620**	13.230	-68.089*	141.424**	-2.820
Dynamism=4	-36.440**	10.826*	-15.383	53.968**	11.225
Dynamism=5	0	0	0	0	0
Management Support=1	39.756**	-7.053	20.433*	-51.588**	-5.439
Management Support=2	39.197**	-5.529	17.141	-49.102**	-11.897
Management Support=3	41.543**	-2.708	20.487*	-49.147**	-6.780
Management Support=4	45.150**	1.354	22.755*	-49.311**	-3.865
Management Support=5	0	0	0	0	0
Project Length	.198**	.129**	.237**	.167**	.014
Organization Size	.081**	-.164**	.047	.229**	-.016
Org. Agile Experience	-.393**	-.230**	.114	-.488**	.268**
Respondent Experience	-.204**	-.148**	-.170**	.125**	-.117**
Respondent Agile Exp.	.545**	-.055	.309	-1.066**	-.076

Table 5.25 presents a summary of the directions of interaction effects, in comparison to the hypothesized direction.

Table 5.25. Cases by Category – Dynamism

	Agile					
	=1	=2	=3	=4	=5	Total
Dynamism=1	0	4	3	3	1	11
Dynamism=2	6	15	20	36	5	82
Dynamism=3	7	34	40	29	9	119
Dynamism=4	4	25	25	19	4	77
Dynamism=5	0	4	6	5	1	16
Total	17	82	94	92	20	305

Table 5.26. Summary of Interaction Effects - Dynamism

Model	Reversed Direction	Not Significant	Hypothesized Direction
Quality	2	0	12
Organizational Benefits	6	7	1
Budget Outcome	3	9	2
Time Outcome	2	0	12
Scope Outcome	1	8	5
Total, (%)	14 (20%)	24 (34%)	32 (46%)

Table 5.27 presents the specific interaction results by model and by discrete category.

Table 5.27. Nested Model Interaction Coefficients – Dynamism

Quality			Agile		
	=1	=2	=3	=4	=5
Dynamism=1	a	36.649**	32.325**	77.307**	0
Dynamism=2	1.517**	76.339**	68.349**	-10.770**	0
Dynamism=3	68.314**	98.264**	49.827**	56.322**	0
Dynamism=4	0	75.967**	36.845**	-9.852**	0
Dynamism=5	a	0	0	0	0
Org Benefits			Agile		
	=1	=2	=3	=4	=5
Dynamism=1	a	-4.533	-5.175	-14.531	0
Dynamism=2	.421*	-13.756	-18.097*	-16.786*	0
Dynamism=3	-2.124	-13.690	-14.856*	-14.755*	0
Dynamism=4	0	-14.736	-11.179*	-13.789*	0
Dynamism=5	a	0	0	0	0
Budget Outcome			Agile		
	=1	=2	=3	=4	=5
Dynamism=1	a	11.729	8.536	36.797	0
Dynamism=2	-1.628**	35.825	23.270	-2.706	0
Dynamism=3	52.554**	56.160*	26.294	45.795**	0
Dynamism=4	0	31.363	15.366	-3.592*	0
Dynamism=5	a	0	0	0	0
Time Outcome			Agile		
	=1	=2	=3	=4	=5
Dynamism=1	a	-48.990**	-32.628**	-102.459**	0
Dynamism=2	3.145**	-94.285**	-82.919**	.871**	0
Dynamism=3	-82.196**	-136.561**	-80.712**	-86.013**	0
Dynamism=4	0a	-100.860**	-50.372**	-1.011**	0
Dynamism=5	a	0	0	0	0
Scope Outcome			Agile		
	=1	=2	=3	=4	=5
Dynamism=1	a	-13.475*	-13.864**	-14.206	0
Dynamism=2	.569*	-17.094	-27.014**	-7.341**	0
Dynamism=3	11.896	-6.990	-7.521	.711	0
Dynamism=4	0	-19.957	-8.287	-6.246**	0
Dynamism=5	a	0	0	0	0

a: no observations, 0: reference group,* p<.05, ** p<.01

The results in Table 5.27 show that the nature of the moderating relationship between the extent of agile method use, dynamism, and quality is as hypothesized.

Summary of results: Hypothesis 2c

Environmental Dynamism will positively moderate the impact of Agile Method Use on Project Success

Significant results in the hypothesized direction for four of the five dimensions of project success. For all of components of project success, the presence of higher levels of dynamism is related to higher impacts of agile method use. While this relationship is not linear, at all levels of agile method use, the as dynamism rises, the model indicates the expectation of more positive expected results for all components of project success.

Summary of results: Hypothesis 1

The Extent of Agile Method Use will positively impact Project Success

We tested the direct effects of the extent of agile method use and the impacts of agile method use in the presence of three moderating variables. In the majority of the models, the results indicated that the impact of the extent of agile method use on project success is positive, although the effects exhibited heterogeneity. On the Quality factor, agile impact use consistently demonstrated positive direct effects. In most models, higher levels of agile method use were associated with higher quality, but these effects were not linear. On the organizational benefits factor, the results were mixed. While agile method use was consistently significant, its effects were extremely nonlinear, with some effects being positive, and some negative, even in the same model.

The project management outcome variables also exhibited direct effects from agile method use, but these results were also mixed. While in the direct effects models, the impacts of agile method use were not significant, but in the moderated models, significant direct effects were found. In the structural complexity tests, all effects were

significant and in the hypothesized direction. In the technical complexity model, the effect on budget outcome was non significant, the effect on time outcome was mixed in direction, and the effect on scope outcome was as hypothesized. Finally, in the dynamism model, the impact on budget outcome was significant, but reversed, while the effect on time outcome and scope outcome was as hypothesized.

In summary, the observed impacts of agile methodologies on project success do, on the whole, support Hypothesis 1. However, the nonlinear relationship between agile method use and project success suggests that the hypothesis is most likely overly simplistic, and requires further development.

Table 5.28 presents a summary of the results of all of the hypothesis tests.

Table 5.28. Summary of Hypothesis Findings

Hypothesis	Product Quality	Org. Benefits	Budget Outcome	Time Outcome	Scope Outcome
H1: Agile Method Use will positively impact project success	Full	Mixed	Mixed	Full	Full
H2a: Structural Complexity will negatively moderate the impact of Agile Method Use	Full	Full	Full	No	Full
H2b: Technical Complexity will negatively moderate the impact of Agile Method Use	No	Full	No	No	No
H2c: Dynamism will negatively moderate the impact of Agile Method Use	Full	Full	Full	Full	Full

Chapter Summary

Significant, nonlinear relationships were found between the theorized explanatory variables and outcome variables. This nonlinearity contributed to mixed results of the hypothesis testing.

Support was found for Hypothesis 1, which stated: “The extent of agile method use will positively impact project success”. Support was found for Hypothesis 2a, which predicted the negative moderation effects of Structural Complexity on the impact of Agile Method Use on Project Success. However, only partial support was found for the impact of Hypothesis 2b, with the hypothesized negative moderating effects of technical complexity only being significant for the organizational benefits component of project success.

Finally, strong support was found for Hypothesis 2c, which stated that dynamism would positively moderate the impact of project success. A significant positive moderating effect was found across all measures of project success.

The results of these hypothesized tests are summarized by project success dimension in Table 5.28

Chapter 6: Discussion and Implications of the Results

Introduction

This chapter discusses the findings, implications, and limitations of this research study. First, the empirical results will be compared with the conceptual models that were proposed and motivated the study. Next, the limitations of the study will be discussed. The chapter will conclude with implications and directions for future research.

The Effectiveness of Agile Methodologies in Software Development

Agile development methodologies have emerged in the past decade as a significant new way of organizing and executing software development projects. Agile practitioners have made numerous claims about the impacts of agile methodologies including associated improvements in team efficiency and performance, higher software quality, and greater organizational benefits. While recent research on agile method use have shown significant results, a number of the normative claims of agile method practitioners have not yet been tested.

Software development practices have long been hypothesized to be contingent upon environmental and other factors (Barki et al. 2001). This means that the performance of a software development method is the product of the interactions between the characteristics of the environment with the characteristics of the development method in use. Agile practitioners have argued that because the software development process is inherently uncertain and empirical, traditional software methodologies have been unable to successfully develop successful plans of execution, even with very high initial investment (Highsmith 2002; Schwaber and Beedle 2002). Instead, agile practitioners have argued that the software development process is much

more like the new product development process than an engineering process (Highsmith 2000). Because of this, agile methodologies have adopted practices, techniques and supporting technologies that are said to enhance delivery success. However, as explained in Chapter 1, many of agile methodologies' normative claims have not been tested, and the methodologies themselves display only partial homogeneity.

In this dissertation we explained that, while the methodologies have diverse enabling practices, and different points of focus, they share a philosophy of the importance of feedback. Whereas previous studies have studied teams performing a particular agile method, we conceptualized agile method use at a level that is measurable across the multiple agile methodologies in practice today, specifically the various feedback processes that are based upon the shared philosophy of the agile manifesto.

The results show that the extent of agile method use positively impacts project success. Significant effects were found on the proposed dimensions of project success, including project management metrics, product quality, and perceived organizational impacts. However, the impact of the extent of agile method use on the project success dimensions was found to be nonlinear. While the effects of agile use were generally positive, the slope of effect is greatly reduced near the mean, and then increases sharply again. This result is intuitive when considering the recommendations of agile practitioners. Agile methodologies advocates argue that the method should be adopted piecemeal, and that a team's greatest pain points should be addressed first. It is therefore likely that there are fast gains from agile methodologies early in the use cycle,

as those practices that are most likely to increase performance are implemented first. However, agile practitioners also stress that, due to the mutually reinforcing nature of the practices of agile, the full performance impact of the method is only achieved when the majority of the practices are put into place. However, the interaction of these practices is complex, and because teams must understand the manner in which to apply them through learning. This means is that after observing the immediate early performance impacts, teams will continue to add agile practices with the anticipation of continued performance gains. However, it should be anticipated that since the "low hanging" fruit has already been addressed, the impacts of these new practices would be lowered. However, once a more complete network of reinforcing practices is put into place, the impact is likely to increase again as the synergistic nature of the practices come into play. This phenomenon requires additional study.

The Moderating Effects of Uncertainty

Hypotheses 2a, 2b, and 2c predicted that three dimensions of uncertainty would moderate the impact of agile methodologies on project success. Uncertainty has been theorized to consist of multiple dimensions of complexity and dynamism. Technical complexity and structural complexity have been explained in the past to be barrier to successful software development. Structural complexity makes the accessing and interpretation of environmental feedback more difficult. Teams within complex structural environments have greater difficulty in establishing goals, interpreting the needs of multiple stakeholders, and in building shared mental models (Xia and Lee 2005). Consistent with prior theory, structural complexity was found to negatively moderate the impact of the extent of agile method use on project success. Structural complexity is

likely to manifest itself through negative impacts on human processes, such as developing shared mental models and processing feedback. Thus, it's presence is intuitively associated with a reduction in the ability of an adaptive team to react to salient cues from the environment.

The moderating impacts of technical complexity on the effects of the extent of agile method use on project success were, in general, insignificant. The impacts of technical complexity were hypothesized to be negative. Significant impacts were found only on the organizational benefits component of project success. This may indicate that the processes of agile methodologies, such as technical feedback and supporting technologies, potentially mitigate the impacts of technical complexity. Further implications of this finding are discussed below.

The third dimension of uncertainty that was hypothesized as moderating the impacts of the extent of agile method use is dynamism. Agile methodologies are designed to manage dynamic environments. Therefore, the use of agile methodologies in a highly dynamic environment is indicative of fit between the environment and the development method in use. We hypothesized that the extent of dynamism in an environment would positively moderate the impacts of agile method use on project success. We proposed a positive interaction between these constructs due to the fact that agile methodologies' feedback processes are designed to recognize and react to environmental change. In the presence of high dynamism, these feedback processes are crucial to project success. However, in the presence of low dynamism, the feedback processes are additional overhead that provide little benefit (Eisenhardt and Tabrizi 1995).

The empirical evidence provided by this study confirms some of the practitioner claims of the impacts of the extent of agile method use on project success. However, the observed relationships between the explanatory and outcome variables was found to be complex. However, this research study has several important limitations.

Limitations of this Study

Because the conceptualization of a generalized extent of agile method use construct was new, and because the impacts of agile on various dimensions of project success had not yet been tested, this research study was exploratory in nature. The first limitation is that the theorized impacts of agile method use were motivated from the perspective of team adaptability and organizational learning. These theories are inherently reinforcing and cyclical in nature, but the research study was performed using a cross-sectional design that indicates the state of the theoretical model at a particular point in time. Learning and adaptation are cyclical processes that occur over time, but cross-sectional research methodologies do not allow researchers to make determinations about rates of change over time, magnitude and changes in magnitude of change over time, and trends. Further, the nature of a survey instrument does not allow for the understanding of specific local variation, but rather simply identifying that the local variation exists. Even so, we believe that cross-sectional research methodologies were appropriate to answer the research question being studied, and to explore the relationship between the use of agile methodologies and project success.

Second, the sample in this study was made up only of agile teams, and was as such as purposive sample. This created intentional sampling bias by including in the research only those teams that self-identified as using agile methodologies. However,

this bias was necessary as the concepts and constructs of the agile team's process and practices would not necessarily be interpretable by non-agile practitioners. This limits the generalizability of the study, and does not indicate the applicability of the study's model or conclusions outside of agile teams.

Third, although the study sought to include the constructs, dimensions, and variables that were indicated by the review of the literature, the final survey that we implemented included less than half of the variables that were initially identified. Even so, the survey took many respondents more than 30 minutes to complete. While respondent participation was encouraged by a small reward, and while respondent fatigue was reduced via a variety of survey design techniques, we could not include all of the variables that the focus groups identified as being salient in agile development. Specifically the nature of interaction with the customer and the frequency and quality of interaction of the team and stakeholders were dimensions that were identified as extremely important by two of the five focus groups. To the extent that the model neglected to measure and consider these variables, we must be cautious in applying the findings.

Implications for Practice

Even today, practitioners look for "silver bullets" with regards to software development methodologies. However, as the prior literature and this study show, the impacts of the use of methodologies such as those designated as agile methodologies is contingent upon fit with the environment. As has been explained, software development methodologies have the greatest impact when there is high fit between environmental factors and the practices of the method. We found this to be the case.

This research reinforces prior research that indicates that structural complexity negatively impacts teams in general. This research shows a significant negative impact on the effects of the extent of agile method use when teams operate in the presence of higher levels of structural complexity. IS practitioners should take care when adopting agile teams to minimize the structural complexity that impacts the teams. When possible, teams should be collocated, and the number of stakeholders and reporting relationships should be minimized.

Also this research found that the impact of technical complexity on success is extremely nonlinear. At both low and high levels of technical complexity, negative interactions between agile use and success were found. This is somewhat contradictory to previous assertions in the practitioner literature that indicate that agile methodologies are best suited for less technically complex projects, and that as technical complexity rises, the need for structure and up front planning rise (Boehm and Turner 2004; Cockburn 2001). One reason for this result may be that if a project is extremely simple, the perceptions of project success and impacts are likely to be low. Conversely, extremely high technical complexity may overwhelm the technical feedback processes of many agile teams. If high system complexity impedes the effect of the technical feedback, the impacts of technical feedback would be mitigated. For instance, if high levels of integration with external systems are necessary, an agile development team may be unable to write tests that fully test the system.

The research suggests that the single most significant component of uncertainty that would indicate a good fit for the use of agile methodologies is the presence of high levels of dynamism. In these environments, the ability to sense and respond to change

has been shown to positively impact performance. However, in environments with lower levels of dynamism, the impacts of agile method adoption and use are not expected to be as high. Organizations should consider the extent to which the team must be able to respond to change as a key indicator signals potentially high performance for agile method use.

In summary, the results of this research indicate that the organizations should generally expect the impacts of agile method use on project success to be generally positive, and to be the most positive in the presence of low structural complexity and high dynamism.

Implications for Future Research

Studying the impacts of agile methodologies remains a rich area for future information systems research. While this research study has been primarily exploratory in nature, it provides a number of new opportunities for future research. This study acts as a useful new data point for the research stream, and provides evidence that more development and empirical research is required.

First, further empirical and theoretical elaboration of the general constructs of agile method use is needed. The most obvious need is to adopt a longitudinal research design that will allow the researcher to investigate the impacts of agile methodologies over time. This would allow the field to better understand whether the impacts of agile and the moderating effects of environmental uncertainty and complexity are consistent over time. Further, the nonlinear nature of the data obtained in this study may imply the presence of a recursive process. A longitudinal or experimental research design would

be able to potentially detect a cyclical or reinforcing effect of agile method use over time.

Second, while we theorized a greater nomological network for the constructs that comprise agile methodologies. While we utilized this theorized nomological network to motivate the hypotheses, the network itself was not tested in this study. Understanding this network would be an interesting extension of this research. Further the nomological network proposed may be generalizable to all software development methodologies. This concept of a nomological network of the components of software development methodologies has not previously been tested. Testing this concept both within the area of agile methodologies, as well as other software development methodologies would be a fruitful addition to the IS field.

Third, the relationships between the constructs are contrary to prior research that propose linear negative relationships between complexity and project success, and linear positive relationships between the extent of agile method use and project success. These findings indicate the need for the development of new theory. Significant theoretical development that recognizes nonlinearity has occurred in the organizational behavior literature. However, IS theories have, with few exceptions, proposed linear relationships, and studies have designed measurement and tests that assume linear relationships. These results add to previous calls for the development of theories of nonlinear effects (Venkatesh and Goyal 2010).

Conclusion

Agile methodologies proceed from the philosophy that the software development effort is one that must consistently and efficiently deal with change. These methodologies prescribe complex networks of processes, practices and procedures, as well as supporting technologies that together are proposed to positively impact the delivery of software in uncertain environments. While these methodologies are extremely heterogeneous in their defined practices, practices that are common to all of the methodologies are those that are designed to elicit feedback cues from the environment. It is only through processing these environmental cues that software development teams can develop adaptability and agility (Burke et al. 2006; Eisenhardt and Tabrizi 1995; Kozlowski 1998). This research focused specifically on the impacts of agile method use, as indicated by the level of use of feedback processes.

Numerous claims have been made about the impacts of agile method use on project success, and the environmental conditions for which the methodologies are most well-suited. Agile method use is expected to have the highest impacts in environments that are highly dynamic. Further, agile method use is most effect when structural complexity barriers are low. Because of the methodologies' focus on human processes of feedback, sense and response high levels of structural complexity impedes the ability of these methodologies to impact the project fully.

The results of this research highlight the importance of feedback in managing the uncertain nature of the software development process. However, the findings indicate a significant need for further research on the impacts of agile methodologies, and environmental conditions on the successful delivery of software development projects.

APPENDICES

APPENDIX A: QUESTIONNARIE ITEMS

Items for Structural Complexity

1. Approximately how many people worked on the project, IN TOTAL? Include everyone who worked on the core project team, even if they only participated in part of the project. *(Scale: Don't Know, 1-5, 6-10, 11-15, 16-20, 21-25, Over 25)*
2. How many companies did members of the project team work for? (i.e., Did everyone work for the same company, or were people paid by multiple companies?) *(Scale: Don't Know, 1, 2, 3, 4, 5, 6+)*
3. How many departments were represented on the project team? *(Scale: Don't Know, 1, 2, 3, 4, 5, 6+)*
4. How many different groups of users provided requirements for the project? *(Scale: Don't Know, 1, 2, 3, 4, 5, 6+)*
5. Please indicate how close or how far away the members of the team were located when working:

Scale:
The entire team worked in the same room/area
The entire team worked in the same building or campus
The entire team worked in multiple locations in the same city
The entire team worked in the same state/province
The entire team worked in the same country
Some of the team worked in a different country
The entire team was spread across different countries

Items for Project Criticality

Scale: 5-point Likert Scale + Don't know (Strongly Disagree – Strongly Agree)

1. The project deliverable was part of the strategic plan of the organization.
2. The project deliverable was required in order to respond to competition.
3. The project deliverable was required in order to respond to government requirements.
4. If the project deliverable was delivered late, it would have significant financial impact to the organization.

Items for Technical Complexity

Scale: 5-point Likert Scale + Don't know (Not Complex At All – Extremely Complex)

1. How technically complex was the organization's system environment?
2. How technically complex was the project system?

Scale: 5-point Likert Scale + Don't know (Strongly Disagree – Strongly Agree)

3. The system involved multiple software environments.
4. The system involved multiple technology platforms.
5. The system involved a lot of integration with other, external systems.
6. The project's integration with external systems was complex.

Items for Dynamism

Scale: 5-point Likert Scale + Don't know (Strongly Disagree – Strongly Agree)

Requirements Dynamism

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

Technical Dynamism

1. The IT infrastructure that the project depended on changed a lot during the project.
2. Software development tools that the project depended on changed a lot during the project.

External Dynamism

1. The project was associated with changes in the users' business processes.
2. The project was associated with changes in the users' organizational structure.
3. The system users' information needs changed a lot during the project.

Items for Agile Method Use

Scale: 5-point Likert Scale + Don't know (Strongly Disagree – Strongly Agree)

Reduced Up Front Planning

1. The team spent less than 10% of the total project timeline on up-front planning (planning that occurred before ANY coding began).
2. At the beginning of the project, the team tried to make only the decisions that were necessary for coding to begin.

Technical Feedback

1. Every programmer was responsible for writing automated tests for the code he or she wrote.
2. Programmers ran a set of automated tests until they all ran successfully before checking in changes.

Environmental Feedback

1. At the beginning of each development cycle, the team and business owners agreed on what would be delivered during the development cycle.
2. The team had a short meeting every day to discuss what was going on that day.
3. The team had a review/verification meeting with stakeholders to demonstrate when software features were complete.
4. On a regular basis, the team reflected on previous work, and looked for ways to improve team performance.
5. At the beginning of each development cycle, the team and business owners agreed on what would be delivered during the development cycle.

Items for Product Quality

Scale: 5-point Likert Scale + Don't know (Definitely Not True – Definitely True)

Prompt: Please give your opinion about the following statements about the project system:

Quality

1. In terms of system quality, I would rate [the project system] highly.
2. Overall, [the project system] is of high quality.
3. I would give the quality of [the project system] a high rating.

Usefulness

1. [The project system] improves users abilities to perform their tasks.
2. The project system] allows users to get work done more effectively.
3. [The project system] allows users to get their tasks done more quickly.

Completeness

1. [The project system] provides users with a complete set features and information.
2. [The project system] is a comprehensive solution.
3. [The project system] provides users with all needed information to do their tasks in the system.

Reliability

1. [The project system] operates reliably.
2. The company can rely on [the project system].
3. The operation of [the project system] is dependable.

Accuracy

1. [The project system] properly performs the tasks it was intended to perform.
2. There are few errors or bugs in [the project system].
3. The information provided by [the project system] is accurate.

Suitability

1. The [the project system] delivered the desired project outcome.
2. The [the project system] accomplishes what was needed.
3. The [the project system] does what was it is supposed to.

Items for Organizational Benefits

Scale: 5-point Likert Scale + Don't know (Definitely Not True – Definitely True)

Prompt: Please give your opinion about these statements related to YOUR PERCEPTION of satisfaction with the project system:

1. Because of this project, our organization can better realize its goals.
2. This project helped our organization to perform better.
3. Our organization is more competitive because of this project.

Items for Project Management Outcomes

Project Budget Outcome

Scale: 5-point Likert Scale + Don't know (Very Much Lower – Very Much Higher)

1. In comparison to the initial budget estimate the final budget was:

Project Time Outcome

Scale: 5-point Likert Scale + Don't know (Very Much Shorter – Very Much Longer)

1. In comparison to the initial time estimate the final project duration was:

Project Scope Outcome

Scale: 5-point Likert Scale + Don't know (Very Much Smaller – Very Much Larger)

APPENDIX B: SUPPLEMENTAL DATA TABLES

Table B.1. Response Breakdown by Team and Response Type

Team ID	Stakeholder	IT Mgmt	Team	Team Total
1	0	0	2	2
2	1	0	0	1
3	0	0	1	1
4	1	0	4	5
5	0	0	1	1
6	4	0	12	16
7	0	0	2	2
8	1	0	5	6
9	1	1	2	4
10	0	1	5	6
11	0	0	4	4
12	0	0	6	6
13	1	0	5	6
14	1	0	2	3
15	0	1	1	2
16	1	0	3	4
17	0	0	5	5
18	1	0	7	8
19	0	0	3	3
20	0	0	4	4
21	1	0	4	5
22	0	0	6	6
23	0	0	4	4
24	1	0	5	6
25	1	1	9	11
26	1	0	10	11
27	0	0	6	6
28	2	0	5	7
29	0	0	1	1
30	0	1	2	3
31	0	1	2	3
32	0	0	1	1
33	0	0	1	1
34	0	0	5	5
35	2	2	5	9
36	1	3	3	7
37	2	2	5	9
38	0	1	3	4

Table B.1. Response Breakdown by Team/Role (cont'd)

Team ID	Stakeholder	IT Mgmt	Team	Team Total
39	1	0	2	3
40	0	1	0	1
41	0	1	0	1
42	0	0	15	15
43	0	1	4	5
44	2	1	11	14
45	3	1	16	20
46	0	0	1	1
47	0	0	1	1
48	0	1	3	4
49	0	0	1	1
50	1	0	3	4
51	1	1	2	4
52	1	0	2	3
53	0	0	1	1
54	0	0	1	1
55	1	0	0	1
56	0	0	1	1
57	0	0	1	1
58	0	0	1	1
59	0	0	1	1
60	0	0	1	1
61	0	0	1	1
62	1	0	0	1
63	1	0	0	1
64	0	0	1	1
65	0	0	1	1
66	0	1	2	3
67	0	1	2	3
68	0	1	5	6
69	1	1	4	6
70	0	1	3	4
71	0	1	2	3
72	1	1	8	10
73	0	1	8	9
74	1	1	7	9
75	2	0	10	12
76	0	1	5	6
77	1	0	5	6
78	0	0	4	4

Table B.1. Response Breakdown by Team/Role (cont'd)

79	0	0	3	3
80	0	0	1	1
81	1	0	0	1
82	0	0	5	5
83	0	0	4	4
Grand Total	42	31	300	373
Average Responses per Team				4.49

REFERENCES

REFERENCES

2009. "Chaos Report," The Standish Group International, Inc., West Yarmouth, MA.

Abrahamsson, P., Salo, O., Ronkainen, J., and Warsta, J. 2002. "Agile Software Development Methods," VTT Technical Research Centre of Finland.

Agerfalk, P., and Fitzgerald, B. 2006. "Old Petunias in New Bowls?," *Communications of the ACM* (49), pp. 10-27.

Ambler, S.W. 2009. "The Agile Scaling Model (Asm): Adapting Agile Methods for Complex Environments." IBM Corporation.

Ancona, D. 1990. "Outward Bound: Strategies for Team Survival in an Organization," *Academy of management journal* (33:2), pp. 334-365.

Ancona, D., and Caldwell, D. 1992. "Bridging the Boundary: External Activity and Performance in Organizational Teams," *Administrative Science Quarterly* (37:4), pp. 634-665.

Anderson, D.J. 2004. *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results*. Prentice Hall.

Argyris, C., and Schoen, D. 1978. *Organizational Learning: A Theory of Action Perspective*. Reading, MA: Addison-Wesley.

Atkinson, R. 1999. "Project Management: Cost, Time and Quality, Two Best Guesses and a Phenomenon, Its Time to Accept Other Success Criteria," *International Journal of Project Management* (17:6), pp. 337-342.

Austin, R., and Devin, L. 2009. "Weighing the Benefits and Costs of Flexibility in Making Software: Toward a Contingency Theory of the Determinants of Development Process Design," *Information Systems Research* (20:3), pp. 462-477.

Avison, D., and Fitzgerald, G. 1998. *Information Systems Development: Methodologies, Techniques and Tools*. Oxford: Blackwell Scientific Publications.

Avison, D., and Taylor, V. 1997. "Information Systems Development Methodologies: A Classification According to Problem Situation," *Journal of Information Technology* (12:1), pp. 73-81.

Baccarini, D. 1996. "The Concept of Project Complexity--a Review," *International Journal of Project Management* (14:4), pp. 201-204.

Barki, H., Rivard, S., and Talbot, J. 2001. "An Integrative Contingency Model of Software Project Risk Management," *Journal of Management Information Systems* (17:4), pp. 37-69.

Beck, K. 1999. *Extreme Programming Explained: Embrace Change*, (First ed.). Addison-Wesley Professional.

Beck, K., and Andres, C. 2004. *Extreme Programming Explained: Embrace Change*, (Second ed.). Addison-Wesley Professional.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. 2001. "Manifesto for Agile Software Development, Accessed July 1, 2011." Retrieved July 1, 2011, 2001, from http://www.agilemanifesto.org

Bhattacharya, S., Krishnan, V., and Mahajan, V. 1998. "Managing New Product Definition in Highly Dynamic Environments," *Management Science*), pp. 50-64.

Boehm, B. 1988. "A Spiral Model of Software Development and Enhancement," *Computer* (21:5), pp. 61-72.

Boehm, B. 2002. "Get Ready for Agile Methods, with Care," *Computer*), pp. 64-69.

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

Bollen, K.A., and Jackman, R.W. 1990. "Regression Diagnostics: An Expository Treatment of Outliers and Influential Cases," in *Modern Methods of Data Analysis,* J. Fox and J. Long (eds.). Newbury Park, CA: Sage, pp. 257-291).

Bovaird, J. 2007. "Multilevel Structural Equation Models for Contextual Factors," in *Modeling Contextual Effects in Longitudinal Studies,* T. Little, J. Bovaird and N. Card (eds.). Mahwah, NJ: Lawrence Erlbaum Associates, pp. 149-182.

Brooks, F. 1987. "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer* (20:4), pp. 10-19.

Brooks, F. 1995. *The Mythical Man-Month (Anniversary Ed.)*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

Burke, C., Stagl, K., Salas, E., Pierce, L., and Kendall, D. 2006. "Understanding Team Adaptation: A Conceptual Analysis and Model," *Journal of Applied Psychology* (91:6), pp. 1189-1207.

Burns, T., and Stalker, G.M. 1994. *The Management of Innovation*. Oxford University Press, USA.

Cao, L., Mohan, K., Xu, P., and Ramesh, B. 2009. "A Framework for Adapting Agile Development Methodologies," *European Journal of Information Systems* (18:4), pp. 332-343.

Coad, P., Lefebvre, E., and DeLuca, J. 1999. *Java Modeling in Color with Uml*. Prentice Hall.

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

Cockburn, A., and Williams, L. 2001. "The Costs and Benefits of Pair Programming." Citeseer, pp. 223-248.

Conboy, K. 2009. "Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development," *Information Systems Research* (20:3), September 1, 2009, pp. 329-354.

Conboy, K., and Fitzgerald, B. 2004. "Toward a Conceptual Framework of Agile Methods," *Extreme Programming and Agile Methods-XP/Agile Universe 2004*), pp. 105-116.

Consortium, D. 2002-2011. "Dsdm Public Version 4.2." Retrieved 1/20, 2011, from http://www.dsdm.org/version4/2/public/

Converse, J.M., and Presser, S. 1986. *Survey Questions: Handcrafting the Standardized Questionnaire*. Sage Publications, Inc.

Creswell, J.W. 2003. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. Sage Publications.

Curtis, B., Krasner, H., and Iscoe, N. 1988. "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM* (31:11), pp. 1268-1287.

DeLone, W.H., and McLean, E.R. 1992. "Information Systems Success: The Quest for the Dependent Variable," *Information Systems Research* (3:1), pp. 60-95.

Delone, W.H., and McLean, E.R. 2003. "The Delone and Mclean Model of Information Systems Success: A Ten-Year Update," *Journal of Management Information Systems* (19:4), pp. 9-30.

DeSanctis, G., and Poole, M.S. 1994. "Capturing the Complexity in Advanced Technology Use: Adaptive Structuration Theory," *Organization Science* (5:2), pp. 121-147.

Dickinson, T.L., and McIntyre, R.M. 1997. "A Conceptual Framework for Teamwork Measurement," in *Team Performance Assessment and Measurement: Theory, Methods, and Applications*. pp. 19-43.

Donaldson, L. 2001. *The Contingency Theory of Organizations*. Thousand Oaks, CA: Sage Publications, Inc.

Duncan, R.B. 1972. "Characteristics of Organizational Environments and Perceived Environmental Uncertainty," *Administrative Science Quarterly* (17:3), pp. 313-327.

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

Eisenhardt, K., and Tabrizi, B.N. 1995. "Accelerating Adaptive Processes: Product Innovation in the Global Computer Industry," *Administrative Science Quarterly* (40:1).

Endsley, M.R. 1995. "Toward a Theory of Situation Awareness in Dynamic Systems," *Human Factors: The Journal of the Human Factors and Ergonomics Society* (37:1), pp. 32-64.

Estublier, J. 2000. "Software Configuration Management: A Roadmap," ACM, pp. 279-289.

Eveleens, J.L., and Verhoef, C. 2009. "The Rise and Fall of the Chaos Report Figures," *IEEE software*), pp. 30-36.

Fowler, M. 1999. *Refactoring. Improving the Design of Exiting Code*. Reading, MA: Addison Wesley Longman.

Fowler, M. 2006. "Continuous Integration." Retrieved March 19th, 2011, 2011, from http://www.martinfowler.com/articles/continuousIntegration.html

Fowler, M., and Highsmith, J. 2001. "The Agile Manifesto," in: *Software Development*. pp. 28-35.

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

Gersick, C. 1988. "Time and Transition in Work Teams: Toward a New Model of Group Development," *Academy of management journal* (31:1), pp. 9-41.

Gersick, C.J.G., and Hackman, J.R. 1990. "Habitual Routines in Task-Performing Groups," *Organizational behavior and human decision processes* (47:1), pp. 65-97.

Group, S. 2009. "Chaos Report, 2009," The Standish Group International, Inc., West Yarmouth, MA.

Harris, M., Collins, R., and Hevner, A. 2009. "Control of Flexible Software Development under Uncertainty," *Information Systems Research* (20:3), pp. 400-419.

Highsmith, J. 2000. *Adaptive Software Development*. Dorset House New York NY.

Highsmith, J. 2002. *Agile Software Development Ecosystems*. Boston: Addison Wesley.

Hinds, P.J., and Mortensen, M. 2005. "Understanding Conflict in Geographically Distributed Teams: The Moderating Effects of Shared Identity, Shared Context, and Spontaneous Communication," *Organization Science*), pp. 290-307.

Hirschheim, R., Klein, H., and Lyytinen, K. 1995. *Information Systems Development and Data Modeling: Conceptual and Philosophical Foundations*. Cambridge Univ Pr.

Ilgen, D., Hollenbeck, J., Johnson, M., and Jundt, D. 2005. "Teams in Organizations: From Input-Process-Output Models to Imoi Models," *Annual Review of Psychology* (56), pp. 517-543.

Indeed.com. 2011. "Agile, Scrum, Extreme Programming, Test Driven Job Trends." Retrieved May 6, 2011, 2011, from http://www.indeed.com/jobtrends?q=agile%2C+scrum%2C+%22extreme+programming%22%2C+%22test+driven%22&l=

Keil, M. 1995. "Pulling the Plug: Software Project Management and the Problem of Project Escalation," *Mis Quarterly* (19:4), pp. 421-447.

Keil, M., Cule, P.E., Lyytinen, K., and Schmidt, R.C. 1998. "A Framework for Identifying Software Project Risks," *Communications of the ACM* (41:11), pp. 76-83.

Keil, M., Mann, J., and Rai, A. 2000. "Why Software Projects Escalate: An Empirical Analysis and Test of Four Theoretical Models," *Mis Quarterly*), pp. 631-664.

Kiesler, S., and Cummings, J.N. 2002. "What Do We Know About Proximity and Distance in Work Groups? A Legacy of Research," *Distributed work* (1), pp. 57-80.

Kirsch, L.J. 1996. "The Management of Complex Tasks in Organizations: Controlling the Systems Development Process," *Organization Science*), pp. 1-21.

Kirsch, L.J., Sambamurthy, V., Ko, D.G., and Purvis, R.L. 2002. "Controlling Information Systems Development Projects: The View from the Client," *Management Science*), pp. 484-498.

Kozlowski, S. 1998. "Training and Developing Adaptive Teams: Theory, Principles, and Research,").

Kozlowski, S., Gully, S., Nason, E., and Smith, E. 1999. "Developing Adaptive Teams: A Theory of Compilation and Performance across Levels and Time," *Pulakos (Eds.), The changing nature of performance: Implications for staffing, motivation, and development*), p. 240ñ292.

Kuchta, D. 2001. "Use of Fuzzy Numbers in Project Risk (Criticality) Assessment," *International Journal of Project Management* (19:5), pp. 305-310.

Lave, J. 1991. *Cognition in Practice*. Cambridge Univ. Pr.

Lawrence, P.R., and Lorsch, J.W. 1967. *Organization and Enviroment*.

Lee, G., and Xia, W. 2010. "Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility," *MIS Quarterly* (34:1), pp. 87-114.

Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., and May, J. 2004. "Agile Software Development in Large Organizations," *Computer*), pp. 26-34.

Louis, M.R., and Sutton, R.I. 1991. "Switching Cognitive Gears: From Habits of Mind to Active Thinking," *Human Relations* (44:1), p. 55.

Lyytinen, K., and Hirschheim, R. 1988. "Information Systems Failuresóa Survey and Classification of the Empirical Literature," Oxford University Press, Inc., pp. 257-309.

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

Mabert, V.A., Soni, A., and Venkataramanan, M. 2003. "The Impact of Organization Size on Enterprise Resource Planning (Erp) Implementations in the Us Manufacturing Sector," *Omega* (31:3), pp. 235-246.

MacCormack, A. 2001. "How Internet Companies Build Software," *MIT Sloan Management Review* (42:2), p. 75ñ84.

MacCormack, A., and Verganti, R. 2003. "Managing the Sources of Uncertainty: Matching Process and Context in Software Development," *Journal of Product Innovation Management* (20:3), pp. 217-232.

Mangalaraj, G., Mahapatra, R.K., and Nerur, S. 2009. "Acceptance of Software Process Innovations - the Case of Extreme Programming," *European Journal of Information Systems* (18:4), pp. 344-354.

Mann, C., and Maurer, F. 2005. "A Case Study on the Impact of Scrum on Overtime and Customer Satisfaction," *Agile Conference, 2005. Proceedings*, pp. 70-79.

Marakas, G.M., and Elam, J.J. 1998. "Semantic Structuring in Analyst Acquisition and Representation of Facts in Requirements Analysis," *Information Systems Research* (9:1), p. 37.

Marks, M.A., and Panzer, F.J. 2004. "The Influence of Team Monitoring on Team Processes and Performance," *Human Performance* (17:1), pp. 25-41.

Maruping, L., Venkatesh, V., and Agarwal, R. 2009a. "A Control Theory Perspective on Agile Methodology Use and Changing User Requirements," *Information Systems Research* (20:3), pp. 377-399.

Maruping, L.M., Zhang, X., and Venkatesh, V. 2009b. "Role of Collective Ownership and Coding Standards in Coordinating Expertise in Software Project Teams," *European Journal of Information Systems* (18:4), pp. 355-371.

McAvoy, J., and Butler, T. 2009. "The Role of Project Management in Ineffective Decision Making within Agile Software Development Projects," *European Journal of Information Systems* (18:4), pp. 372-383.

McConnell, S. 1996. *Rapid Development: Taming Wild Software Schedules*. Microsoft Press Redmond, WA, USA.

McGrath, J. 1964. *Social Psychology: A Brief Introduction*. Holt, Rinehart and Winston.

McGrath, J., Arrow, H., and Berdahl, J. 2000. "The Study of Groups: Past, Present, and Future," *Personality and Social Psychology Review* (4:1), pp. 95-105.

McIntyre, R.M., and Salas, E. 1995. "Measuring and Managing for Team Performance: Emerging Principles from Complex Environments," in *Team Effectiveness and Decision Making in Organizations*. pp. 9-45.

McKeen, J.D., Guimaraes, T., and Wetherbe, J.C. 1994. "The Relationship between User Participation and User Satisfaction: An Investigation of Four Contingency Factors," *MIS Quarterly*), pp. 427-451.

Meyer, M.H., and Curley, K.F. 1991. "An Applied Framework for Classifying the Complexity of Knowledge-Based Systems," *Mis Quarterly*), pp. 455-472.

Miller, D. 1992. "Environmental Fit Versus Internal Fit," *Organization Science* (3:2), pp. 159-178.

Moran, D. 2010. "Top 10 Reasons to Use Agile Development." Retrieved 3/9/2011, 2011, from http://www.devx.com/enterprise/Article/44619

Nerur, S., Mahapatra, R.K., and Mangalaraj, G. 2005. "Challenges of Migrating to Agile Methodologies," *Communications of the ACM* (48:5), pp. 72-78.

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

Ogunnaike, B.A., and Ray, W.H. 1994. *Process Dynamics, Modeling, and Control*. Oxford University Press New York:.

Okhuysen, G.A. 2001. "Structuring Change: Familiarity and Formal Interventions in Problem-Solving Groups," *The Academy of Management Journal* (44:4), pp. 794-808.

Okhuysen, G.A., and Waller, M.J. 2002. "Focusing on Midpoint Transitions: An Analysis of Boundary Conditions," *The Academy of Management Journal* (45:5), pp. 1056-1065.

One, V. 2010. "5th Annual State of Agile Development Survey Final Summary Report."

Orlikowski, W.J. 1992. "The Duality of Technology: Rethinking the Concept of Technology in Organizations," *Organization Science* (3:3), pp. 398-427.

Parrish, A., Smith, R., Hale, D., and Hale, J. 2004. "A Field Study of Developer Pairs: Productivity Impacts and Implications," *Software, IEEE* (21:5), pp. 76-79.

Paulk, M., Weber, C., Curtis, B., and Chrissis, M. 1995. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Indianapolis, IN: Addison Wesley.

Petter, S., DeLone, W., and McLean, E. 2008. "Measuring Information Systems Success: Models, Dimensions, Measures, and Interrelationships," *European Journal of Information Systems* (17:3), pp. 236-263.

Pfeffer, J., and Salancik, G.R. 2003. *The External Control of Organizations: A Resource Dependence Perspective*. Stanford, CA: Stanford University Press.

Phillips, J.J., Bothell, T.W., and Snead, G.L. 2002. *The Project Management Scorecard: Measuring the Success of Project Management Solutions*. Butterworth-Heinemann.

Pich, M.T., Loch, C.H., and De Meyer, A. 2002. "On Uncertainty, Ambiguity, and Complexity in Project Management," *Management Science*), pp. 1008-1023.

PMBOK. 2000. *A Guide to the Project Management Body of Knowledge (Pmbok Guide)*. Project Management Institute.

Port, D., and Bui, T. 2009. "Simulating Mixed Agile and Plan-Based Requirements Prioritization Strategies: Proof-of-Concept and Practical Implications," *European Journal of Information Systems* (18:4), pp. 317-331.

Purvis, R.L., Sambamurthy, V., and Zmud, R.W. 2001. "The Assimilation of Knowledge Platforms in Organizations: An Empirical Investigation," *Organization Science*), pp. 117-135.

Rai, A., Lang, S.S., and Welker, R.B. 2002. "Assessing the Validity of Is Success Models: An Empirical Test and Theoretical Analysis," *Information Systems Research* (13:1), p. 50.

Ribbers, P.M.A., and Schoo, K.C. 2002. "Program Management and Complexity of Erp Implementations," *Engineering Management Journal* (14:2), pp. 45-52.

Salas, E., Cannon-Bowers, J., Fiore, S., and Stout, R. 2001. "Cue-Recognition Training to Enhance Team Situation Awareness," *New trends in cooperative activities: understanding system dynamics in complex environments. Santa Monica, CA: Human Factors and Ergonomics Society*), p. 169ñ190.

Salas, E., Prince, C., Baker, D.P., and Shrestha, L. 1995. "Situation Awareness in Team Performance: Implications for Measurement and Training," *Human Factors: The Journal of the Human Factors and Ergonomics Society* (37:1), pp. 123-136.

Salo, O., and Abrahamsson, P. 2008. "Agile Methods in European Embedded Software Development Organisations: A Survey on the Actual Use and Usefulness of Extreme Programming and Scrum," *Software, IET* (2:1), pp. 58-64.

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

Schwaber, C., and Fichera, R. 2005. "Corporate It Leads the Second Wave of Agile Adoption," Cambridge, MA.

Schwaber, K. 1996. "Controlled Chaos: Living on the Edge," *American Programmer* (9), pp. 10-16.

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

Seddon, P.B. 1997. "A Respecification and Extension of the Delone and Mclean Model of Is Success," *Information Systems Research* (8:3), pp. 240-253.

Shenhar, A.J., and Dvir, D. 1996. "Toward a Typological Theory of Project Management," *Research Policy* (25:4), pp. 607-632.

Stephens, M., and Rosenberg, D. 2003. *Extreme Programming Refactored: The Case against Xp.* New York, NY: Apress.

Tait, P., and Vessey, I. 1988. "The Effect of User Involvement on System Success: A Contingency Approach," *Mis Quarterly*), pp. 91-108.

Tesluk, P.E., and Mathieu, J.E. 1999. "Overcoming Roadblocks to Effectiveness: Incorporating Management of Performance Barriers into Models of Work Group Effectiveness," *Journal of Applied Psychology* (84:2), p. 200.

Thompson, J. 2003. *Organizations in Action: Social Science Bases of Administrative Theory*. Transaction Pub.

Turner, J.R., and Cochrane, R.A. 1993. "Goals-and-Methods Matrix: Coping with Projects with Ill Defined Goals and/or Methods of Achieving Them," *International Journal of Project Management* (11:2), pp. 93-102.

Venkatesh, V., and Goyal, S. 2010. "Expectation Disconfirmation and Technology Adoption: Polynomial Modeling and Response Surface Analysis," *Mis Quarterly* (34:2), pp. 281-303.

Venkatraman, N. 1989. "The Concept of Fit in Strategy Research: Toward Verbal and Statistical Correspondence," *Academy of Management Review*), pp. 423-444.

Wallace, L., and Keil, M. 2004. "Software Project Risks and Their Effect on Outcomes," *Communications of the ACM* (47:4), pp. 68-73.

Waller, M.J. 1999. "The Timing of Adaptive Group Responses to Nonroutine Events," *The Academy of Management Journal* (42:2), pp. 127-137.

Weiss, D.J. 1986. "The Discriminating Power of Ordinal Data," *Journal of Social Behavior and Personality* (1:38), pp. 1-389.

West, D., and Grant, T. 2010. "Agile Development: Mainstream Adoption Has Changed Agility," Forrester Research.

Williams, L., Kessler, R.R., Cunningham, W., and Jeffries, R. 2000. "Strengthening the Case for Pair Programming," *Software, IEEE* (17:4), pp. 19-25.

Williams, T.M. 1999. "The Need for New Paradigms for Complex Projects," *International Journal of Project Management* (17:5), pp. 269-273.

Wixom, B.H., and Todd, P.A. 2005. "A Theoretical Integration of User Satisfaction and Technology Acceptance," *Information Systems Research* (16:1), pp. 85-102.

Wynekoop, J., and Russo, N. 1995. "Systems Development Methodologies: Unanswered Questions," *Journal of Information Technology* (10:2), pp. 65-73.

Xia, W., and Lee, G. 2005. "Complexity of Information Systems Development Projects: Conceptualization and Measurement Development," *Journal of Management Information Systems* (22:1), pp. 45-83.

Zmud, R.W., Anthony, W.P., and Stair Jr, R.M. 1993. "The Use of Mental Imagery to Facilitate Information Identification in Requirements Analysis," *Journal of Management Information Systems* (9:4), pp. 175-191.