Journal of the Association for Information Systems

Volume 17 | Issue 4 Article 1

4-28-2016

Job Satisfaction in Agile Development Teams: Agile Development as Work Redesign

John F. Tripp
Baylor University, john_tripp@baylor.edu

Cindy Riemenschneider
Baylor University, C_Riemenschneider@baylor.edu

Jason B. Thatcher
Clemson University, jason.b.thatcher@gmail.com

Recommended Citation
Tripp, John F.; Riemenschneider, Cindy; and Thatcher, Jason B. (2016) "Job Satisfaction in Agile Development Teams: Agile Development as Work Redesign," *Journal of the Association for Information Systems*, 17(4), .
DOI: 10.17705/1jais.00426
Available at: https://aisel.aisnet.org/jais/vol17/iss4/1

Journal of the Association for Information Systems

Research Paper ISSN: 1536-9323

Job Satisfaction in Agile Development Teams: Agile Development as Work Redesign

John F. Tripp
Information Systems Department, Baylor University
John_Tripp@baylor.edu

Cindy Riemenschneider
Information Systems Department, Baylor University
C_Riemenschneider@baylor.edu

Jason B. Thatcher
Department of Management, Clemson University
jason.b.thatcher@gmail.com

Abstract:

Agile software-development advocates claim that an important value proposition of agile methods is that they make people more motivated and satisfied with their jobs. While several studies present anecdotal evidence that agile methods increase motivation and satisfaction, research has not theoretically explained or empirically examined how agile development practices relate to team members' feelings about their work. Drawing on the management and software-development literature, we articulate a model of job design that connects agile development practices to perceptions of job characteristics and, thereby, improve agile team members' job satisfaction. Using data collected from 252 software-development professionals, we tested the model and found a positive relationship between agile project-management and software-development practices and employees' perceptions of job characteristics. Further, we found direct effects between agile development-practice use and job satisfaction. Finally, we found interaction effects between the use of agile project-management and software-development approaches and the perception of job autonomy. With this study, we contribute to the literature by theoretically explaining and directly evaluating agile development practices' impact on individuals' perceptions about their job characteristics and on their job satisfaction.

Keywords: Agile Development, Job Satisfaction, Job Characteristics Model.

Fred Niederman was the accepting senior editor. This paper was submitted on November 5, 2014, and went through three revisions.

1 Introduction

Over the last decade, a growing number of companies have employed agile software-development methods (West, Grant, Gerush, & D'Silva, 2010). Regardless of whether they advocate for agile methods such as scrum, extreme programming (XP), crystal clear, feature-driven development (FDD), or kanban, agile-method advocates make two overarching claims. First, they claim that agile methods produce better software—a claim that research that examines project success, software quality, and other measures has supported (e.g., Maruping, Venkatesh, & Agarwal, 2009a; Maruping, Zhang, & Venkatesh, 2009b; Nevo & Chengalur-Smith, 2011). Second, they claim that using agile methods creates more satisfied employees (Highsmith, 2002)—a claim that researchers have rarely empirically investigated.

Limited research connects agile development (i.e., using agile methods and their associated practices) to software-development team members' satisfaction. One study of software developers found that satisfaction was higher with pair programming (an agile development practice) than with traditional individual programming (Balijepally, Mahapatra, Nerur, & Price, 2009). These authors speculate that "higher satisfaction of programming pairs has implications for reducing employee turnover" (Balijepally et al., 2009, p. 111). In a study of teams, another group of authors investigated the relationship between using agile practices and team motivation (McHugh, Conboy, & Lang, 2011). They found that three agile practices (i.e., iteration planning, daily stand-up meetings, and iteration retrospectives) were associated with job characteristics such as job autonomy, skill variety, and feedback. Similarly, Tessem and Maurer (2007) found that some members of a particular agile team had higher perceptions of job characteristics and job satisfaction but others did not. These and other studies of agile development methods provide evidence of their impact on work-related perceptions including job satisfaction, but they have substantial theoretical and methodological limitations (e.g., Layman, Williams, & Cunningham, 2004; Mannaro, Melis, & Marchesi, 2004; Mann & Maurer, 2005; McHugh, Conboy, & Lang, 2010).

For instance, some researchers based their findings on a single team or a few teams (e.g., McHugh & Lang, 2011; Tessem & Maurer, 2007). Furthermore, many researchers investigated the influence of one or only a few of the practices enumerated in an agile method. For instance, McHugh et al. (2012) focused on three practices from the scrum method (sprint/iteration planning, daily stand-ups, and sprint/iteration retrospectives) in investigating trust in agile teams. Tessem and Maurer (2007) used a single agile team that employed the scrum method to investigate the impact of agile methods on team members' job satisfaction. Although Melnik and Maurer (2006) drew a large sample (n = 448) to compare job satisfaction of agile development and traditional team members, their study was descriptive, failed to measure the use of agile methods, and did not test a mechanism for how agile methods impact job satisfaction. Consequently, while some evidence suggests an agile method-satisfaction relationship, it is important to conduct a rigorous, theory-based field study of the relationship between agile development methods and job satisfaction.

We need to realize a deeper understanding of the relationship between agile development methods and job satisfaction because research in information systems (IS), marketing, and management consistently suggests that job design is related to employee satisfaction, which results in positive outcomes for employing organizations. A meta-analysis of studies on employee attitudes found that job design influences job satisfaction, which, in turn, influences work performance (Harrison, Newman, & Roth, 2006). In information technology (IT) workforce research, a substantial body of research connects job design with outcomes such as intrinsic motivation, job satisfaction, and turnover intention (Joseph, Ng, Koh, & Ang, 2007; Thatcher, Liu, Stepina, Goodman, & Treadway, 2006; Thatcher, Stepina, & Boyle, 2002). Because agile methods and their associated practices change the structure of software developers' work (Tessem & Maurer, 2007)—for example, in pair programming (Cockburn & Williams, 2001), developing tests first (Beck, 2003), and delivering iteratively (Moe, Dingsoyr, & Dyba, 2010)—we propose that the use of these methods affects employees' motivation and satisfaction.

In this paper, we investigate the relationship between the use of agile methods and agile team members' job satisfaction. Although agile-method proponents claim a common philosophy as described in the agile manifesto (Fowler & Highsmith, 2001), each method emerged out of different circumstances and has a different focus. For instance, scrum (Schwaber & Beedle, 2002) is primarily a project management method, while extreme programming (XP) (Beck, 2000) focuses more on software-development concerns. Because of their different heritage, each method prescribes different and sometimes contradictory practices. For example, while XP prescribes collective (team) code ownership, feature-driven development (FDD) (Palmer & Felsing, 2002) prescribes that each code component have a defined, single owner. We apply the term "practices" to specific agile techniques, such as pair programming and collective code ownership. In contrast, “methods” refers to any of a number of defined, interdependent sets of practices developed by software practitioners, such as XP, FDD, and scrum.

In addition, although agile practitioners claim to adopt methods, few adopt all of the practices in a method (Conboy & Fitzgerald, 2010), and many adopt practices from multiple methods (VersionOne, 2011). Because of this variance in adoption, even among teams that claim to adopt the same method, we primarily focus on the use of specific agile practices and their impact on job satisfaction in this paper. In doing so, we pursue two research questions. First, proponents argue that agile method influences job satisfaction because of the combined and overlapping effect of multiple practices (Beck, 2000; Highsmith, 2002). These practices may not contribute equally, or at all, to job satisfaction. Hence, we examine the following research question:

RQ1: How does the extent of use of different agile practices impact job satisfaction?

To explore this question, we developed a theoretical model that links agile practices with agile-development team members’ motivation and job satisfaction. Initial research on the use of agile methods, motivation, and job satisfaction has found that agile team members are more motivated (McHugh & Lang, 2011) and satisfied than traditional-development team members (Melnik & Maurer, 2004). A common thread through this literature is that agile practices affect team members’ perceptions of job characteristics, a key predictor of job satisfaction (Hackman & Oldham, 1980). We leverage the existing body of knowledge on job design and IT workforce management (e.g., Ahuja, Chudoba, Kacmar, McKnight, & George, 2007; Moore, 2000; Morris & Venkatesh, 2010; Thatcher et al., 2002) to motivate a model that connects the use of popular agile practices and processes to job satisfaction, both directly and indirectly, through their effect on perceptions of job characteristics. Hence, we developed a model that investigates our second research question:

RQ2: What are the causal pathways through which agile practices affect job satisfaction?

Throughout this paper, we develop and test a model that evaluates agile practices’ impact on job satisfaction and how agile method use impacts employee perceptions of job characteristics. This model contributes to the agile development and IS literature by 1) investigating the antecedents of job perceptions, 2) recognizing that agile methods and their defined practices are largely instances of job design, and 3) developing a theoretical lens through which to explain the impacts of agile methods on job attitudes. In Section 2, we address the theoretical background for this study.

2 What is an Agile Development Team?

Agile development teams (ADT) are groups of people that work together to build new software systems and modify existing software systems. ADTs usually include members with a broad skill set, such as analysis, programming, design, database architecture and administration, systems engineering, and project management. In fact, agile teams share the premise that everyone who the team needs to fill all the roles necessary to complete the project must participate in the project team (Highsmith, 2002). This situation differs from the traditional waterfall/structured environments, where teams often specialize according to function (e.g., analysis, design, development, testing) (Nerur, Mahapatra, & Mangalaraj, 2005)[1].

ADT team members often have different responsibilities and titles across organizations. For example, systems analysts might perform programming and design responsibilities in their job even though their job does not inherently delineate those responsibilities. Or a developer might perform programming, testing, and configuration management tasks that are fragmented in a traditional setting. Therefore, we use the term ADT to represent the plethora of job titles and roles fulfilled by the team delivering a software product.

ADTs use an agile method such as scrum, XP, or kanban or combinations of practices from multiple agile methods (Highsmith, 2002; VersionOne, 2011). ADTs typically work in short cycles in which they deliver working software incrementally and iteratively and focus on emergently understanding the problem space during development (Beck, 2003; Cockburn, 2001; Harris, Collins, & Hevner, 2009; Schwaber & Beedle, 2002). In contrast to traditional teams, agile teams do not focus on specifying system requirements and design up-front or fully understanding the problem space early on (Boehm, 1984) because they lack the longer delivery cycles employed by traditional project-management techniques (Boehm & Turner, 2004; Highsmith, 2002).

[1] We use the terms “traditional” and “waterfall” or “structured” as a shortcut for referring to traditional development approaches but realize that “waterfall” as such is not necessarily fully implemented, nor does it preclude some level of iteration in many organizations.

Even though many agile methods call for heavy customer involvement with the ADT (e.g., XP's on-site customer practice)—even in some cases calling for the customer to be part of the ADT—we do not include customers in our conceptualization of the ADT for two reasons. First, not all ADTs include the customer as a part of the team. Although all agile methods call for a product owner's regular availability for decision making, it is unusual for most ADT customers to be full-time members of the team (Cockburn, 2001; Highsmith, 2002). Second, agile proponents' claims regarding job satisfaction revolve around ADT members who are part of the core team delivering the software (Beck, 2000; Highsmith, 2002). As such, our theory development focuses on ADT roles that are typically part of the information technology function in organizations: software developers, quality assurance and test analysts, business analysts, software architects, project managers, and team leads.

In summary, an ADT 1) is a cross-functional team that builds and updates software; 2) predominantly uses agile management and development practices, such as iterative delivery or pair programming; and 3) comprises members whose responsibilities focus on delivering software rather than externally managing the team or business.

2.1 The Job Characteristics Model and Agile Development Teams

By employing the practices defined in agile methods, organizations change the design of ADT members' work. For example, agile methods require software developers to participate in activities beyond programming such as systems analysis and project management, which will likely increase their level of perceived job autonomy and skill or task variety. Furthermore, the management and IS literature has suggested that, by changing job characteristics, agile methods might impact ADT members' motivation, job satisfaction, and work performance.

The job characteristics model (JCM) (Hackman & Oldham, 1980; Joseph et al., 2007) represents a useful lens for understanding how changes wrought by agile methods affect job satisfaction. The JCM identifies five job characteristics that influence workers' perceptions of the job and their attitudes about it: 1) task significance, which the extent to which people believe that their job impacts the lives of people—either society at large or in an organization; 2) task identity, which is the extent to which a job's tasks are "whole" or involve the completion of an identifiable outcome; 3) skill variety, which is the extent to which one perceives a job as requiring a variety of skills, talents, and experiences; 4) job autonomy, which is the extent to which an employee has the discretion about how to complete the work required and set a schedule for completion; and 5) feedback, which is the extent to which the process of completing the work provides employees with information through which they can evaluate their own performance.

Research has used the JCM to explain IT professionals' job satisfaction (e.g., Morris & Venkatesh, 2010), turnover intention (e.g., Thatcher et al., 2002), and work exhaustion (e.g., Moore, 2000), among other outcomes. Job satisfaction is an affective response resulting from work experiences (Weiss & Cropanzano, 1996), whereas work exhaustion is the depletion of emotional, mental, and physical resources because of work experiences (Moore, 2000). Research has shown work exhaustion to be associated with lower job satisfaction (Burke & Greenglass, 1995; Rutner, Hardgrave, & McKnight, 2008) and higher turnover (Moore, 2000). Job characteristics are strongly related to job satisfaction and work exhaustion (e.g., Ahuja et al., 2007; Ang & Slaughter, 2001; Moore, 2000; Morris & Venkatesh, 2010; Rutner et al., 2008; Thatcher et al., 2002). While studies on turnover in broad populations of IT professionals (Dinger, Thatcher, Treadway, Stepina, & Breland, 2015; Moore, 2000; Thatcher et al., 2006) and more narrowly defined populations such as IT road warriors (Ahuja et al., 2007) and software developers (Ply, Moore, Williams, & Thatcher, 2012) have found the JCM to be useful to explain job satisfaction, we lack rigorous research on how agile methods impact employee perceptions of job characteristics or job satisfaction is scarce (for an exception, see Pedrycz, Russo, & Succi, 2011).

Hence, to examine agile practitioners' claims about their methods' resulting in higher job satisfaction, we use the JCM as a lens through which to theorize about the impacts of agile software-development methods on ADT members' perceptions of and satisfaction with their jobs. Table 1 presents the definitions of the job characteristics and the other constructs we use in this study.

Table 1. Constructs Used in this Study

Construct name	Definition (source)
Dependent variable	
Job satisfaction	The extent of positive emotional response to the job resulting from an employee's appraisal of the job as fulfilling or congruent with the individual's values (Morris & Venkatesh, 2010; Weiss & Cropanzano, 1996).
Perceptions of job characteristics	
Job autonomy	The extent to which a job provides employees with discretion to choose how they do work and to set the schedule for completing the work activities (Hackman & Oldham, 1980; Moore, 2000; Thatcher et al., 2002).
Feedback	The extent to which carrying out the work activities provides employees with clear information about their own performance (Hackman & Oldham, 1980; Moore, 2000; Thatcher et al., 2002).
Skill variety	The extent to which a job requires the use of different talents (JHackman & Oldham, 1980; Moore, 2000; Thatcher et al., 2002).
Task identity	The extent to which a job involves completing a whole identifiable outcome (Hackman & Oldham, 1980; Moore, 2000; Thatcher et al., 2002).
Task significance	The extent to which a job has impact on the lives of people in an organization or society in general (Hackman & Oldham, 1980; Moore, 2000; Thatcher et al., 2002).
Independent variables	
Agile project-management practices	The extent to which the respondent's team uses the agile project-management practices defined in this study; namely, burndown, daily stand-ups, retrospective meetings, and iterative delivery.
Agile software-development approach management practice	The extent to which the respondent's team uses the agile software-development approach practices defined in this study; namely, pair programming, automated builds, continuous integration, unit testing, refactoring, code standards.

We contend that ADTs organize work in a fashion that results in their members' more positively perceiving job characteristics. ADTs do so by instantiating Hackman and Oldham's (1980) five job-design principles that lead to higher perceptions of job characteristics and job satisfaction: 1) combining tasks, 2) forming natural work units, 3) establishing client relationships, 4) vertically loading the job, and 5) opening feedback channels (Figure 1). ADT proponents argue that the principles of scientific management, which traditional software-development practices embody, fractionalize work, lower perceived job characteristics, and diminish motivation and satisfaction.

Agile practices might de-fractionalize software-development processes by requiring team members to combine skills and tasks and share responsibilities; doing so results in improved job-characteristic perceptions and related attitudes such as job satisfaction. For example, rather than taking an assembly line approach in which each worker performs a small part of the work and then hands it off to the next worker, agile methods require that each ADT member assume responsibility for building software. In this way, agile methods help ADT members to better understand the scope and significance of the work being performed (Hackman & Oldham, 1980). In Section 2.2, we describe each of the job-design principles and map specific agile practices to the job-design principles (see Figure 1), which will lead to improved job characteristics perceptions.

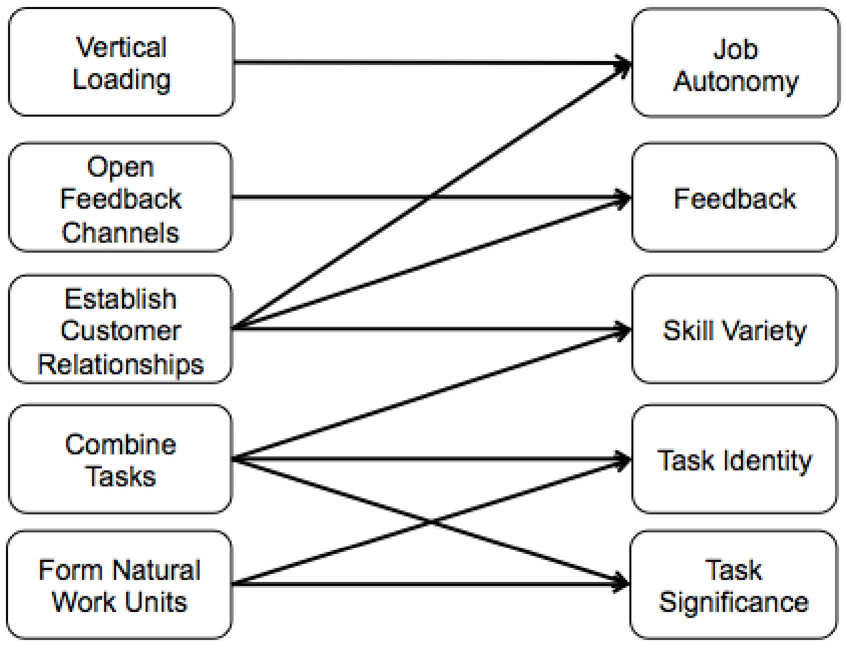


Figure 1. Job-design Principles as Related to Job Characteristics (Hackman & Oldham, 1980)

2.2 Agile Methods as Job Redesign

Agile methods prescribe a wide range of practices that can affect perceived job characteristics. We focus on practices rather than methods because agile teams consider methods to be a toolkit that they can draw on as needed (Conboy & Fitzgerald, 2010). Although some teams self-identify as using one particular method or another, a large proportion of teams use hybrid methods comprising practices from multiple methods (VersionOne, 2011). Although developers in agile development environments use at least 25 practices, more than half of these environments adopt only 11 (VersionOne, 2011). Because of their high reported adoption, we include these 11 practices in our review of agile and job design. Additionally, we also include pair programming (rank order 16) because of its reported influence on job characteristics (e.g., Tessem & Maurer, 2007). Table A3 provides the rank order and the usage percentage of these 12 and other agile practices.

We offer succinct definitions of the agile practices we investigate in this study (see Table 2). One can place agile development practices into two primary categories (Tripp & Armstrong, 2014)[2]. The first relates to project management and includes daily stand-ups, iterative delivery, retrospectives, and burndowns. The second relates to the ADT's software-development approach (SDA) and includes automated (unit) testing, automated builds, continuous integration, coding standards, and refactoring. As we describe above, due to the evidence as to its potential impact on job satisfaction, we include pair programming in the agile SDA category (Table 2). We propose that these agile practices instantiate job-design principles as Table 3 summarizes. We describe these proposed impacts of the use of each agile practice on job-design principles starting with combining tasks, followed by forming natural work units, establishing client relationships, vertically loading the job, and opening feedback channels. First, we describe each job-design principle. Second, we provide a specific illustration of the applicable project management or software development practices associated with that job-design principle.

[2] Tripp and Armstrong performed a two-round categorization exercise for the 12 agile practices using three expert agile developers to categorize the practices. The judges identified two categories: agile project management and agile software-development approach. The judges had 97.2 percent agreement on categorization.

Table 2. Definitions of Agile Practices Described in this Study

Agile practice name	Definition (source)
Project-management practices	
Daily stand-up meeting (Schwaber & Beedle, 2002)	A (usually) daily meeting in which all project participants meet while standing to encourage brevity. In scrum, the meeting involves asking and answering three questions*. 1. What did I accomplish yesterday? 2. What will I do today? 3. What obstacles are impeding my progress? A kanban daily stand-up asks and answers: 1. What obstacles are impeding my progress? 2. What has progressed? * The meeting focuses on these questions as much as is practical.
Iterative delivery Release planning Iteration planning Velocity	The process of 1) planning and 2) delivering in an incremental manner. Specifically, the concept that delivery in small chunks provides the team with the ability to generate code and immediately receive feedback from the environment after each iteration. Velocity emerges after the team has delivered several iterations and measures the amount of work that it can complete per iteration.
Retrospectives (Schwaber & Beedle, 2002)	A meeting held at the end of each iteration in which the team 1) critically reflects on the last iteration and 2) identifies and implements continuous improvement opportunities.
Burndown	A visual representation that provides information to the team about 1) work that has been completed and 2) work remaining to be completed in the current iteration or release.
Software-development approach practices	
Automated (unit) testing (Beck, 2000)	Using dedicated test code that one can run (usually automatically) to test the effects of changes to the system. The team generally performs this testing before ADT members are allowed to check in code, which allows developers to be sure that they have not broken anything in the system.
Automated builds (Küster, Gschwind, & Zimmermann, 2009)	Using a code script to rebuild the software, which allows the team to be sure that any files necessary to successfully build the software have been added to the code repository.
Continuous integration (Beck, 2000; Duvall, Matyas, & Glover, 2007)	The process of systematically and regularly building and deploying the code to a test server.
Coding standards (Beck, 2000)	A set of established norms as to code-naming and consistency.
Refactoring (Fowler, 1999)	A commitment by the team to use practices that lead to removing redundancy, eliminating unused functionality, and refreshing obsolete designs.
Pair programming (Beck, 2000; Cockburn & Williams, 2001)	The practice of two developers working together to develop a portion of code.

Table 3. Job-design Principles and Agile Practices

Agile practice	CT	FNWU	ECR	VL	OFC
Project-management practices					
Daily stand-up meeting			x		x
Iterative delivery Release planning Iteration planning Velocity		x	x	x	
Retrospectives				x	x
Burndown					x
Software-development approach					
Automated (unit) testing	x				x
Automated builds	x				x
Continuous integration					x
Coding standards	x				
Refactoring	x			x	
Pair programming	x				x
CT: combining tasks, FNWU: form natural work units, ECR: establish client relationships, VL: vertical job loading, OFC: opening feedback channels					

2.3 Combining Tasks

Properly developing code requires software-development teams to write code, test code, and ensure that one can deploy the code as written and tested. Traditional software-development takes a work-fractionalization approach. Work is divided across software-development teams who focus on particular layers of the software (database, business logic, user interface) or phases of the project lifecycle. For example, activities such as coding, testing, and deploying software are often segmented into different phases of the project lifecycle and completed by three distinct teams. By dividing the work across teams, foci, and phases of the lifecycle, traditional development methods reduce software developers' ability to see the results of tasks related to a completed product.

Agile teams organize work in a significantly different fashion. ADTs recombine the responsibilities of software developers to include multiple tasks and the development of multiple layers of the software, which leads to better understanding of the complete system and not just development of a particular component (Anderson, 2004). Because ADTs desire to ensure that one can deploy their code after each iteration, this entire process occurs continuously. We propose that four of the selected agile practices relate to the perception of recombining tasks and higher perceived job characteristics (Table 2, column 2).

2.3.1 Automated (Unit) Testing and Combining Tasks

Although developers in traditional software environments typically test software before deployment, most traditional teams do not write persistent shared code suites to test software systems. Instead, each developer might perform initial unit testing manually or with some ad-hoc code written for testing their particular changes. Further, in traditional teams, a dedicated team of system testers typically fully tests systems at a later time.

When using automated-testing frameworks, ADTs write both unit and system tests that they preserve so all members of the ADT can use them. These tests combine tasks in two ways. First, the code is an example of documentation in practice (Beck, 2000; Küster et al., 2009) because the tests themselves provide guidance as to the system's functionality. Second, one can run these tests at any time, which allows a developer to test the consequences of changing code across the entire system. In this way, unit tests combine tasks previously performed by the business analyst role and the tester role.

2.3.2 Automated Builds and Combining Tasks

Automated builds refer to the process of creating scripts to generate a complete and deployable build, and many ADTs require that team members run automated builds before they check in code changes (Küster et al., 2009). When ADTs do this, a developer adopts tasks beyond the traditional boundaries of the role and performs the tasks of a configuration manager. Automated builds help to ensure that a developer has included and configured all dependencies required to deploy the software.

This process vertically loads the ADT by providing control over execution of scripts, which reduces the need for a dedicated configuration manager role, and places the responsibility for ensuring that the packages are complete and deployable onto each ADT member. Further, ADTs run the automated build processes regularly, most often before a developer submits code (Küster et al., 2009). During this process, the developer receives a notification about whether their new changes have broken any other portions of the code base and whether one can properly build the code on their machine. As such, automated builds provide feedback directly to the developers as they complete code.

2.3.3 Coding Standards, Refactoring, and Combining Tasks

The term coding standards refers to the ADT's ensuring that it follows agreed-on guidelines when coding. As such, ADT developers combine tasks by ensuring that they do not simply code software that works but software that is easier to refactor and easier to assign to others to work on (Auer & Miller, 2002). Whereas software code adopts a format according to multiple individual preferences, coding standards require developers to ensure that the code itself is not a personal object but a community artifact (Beck, 2000). As such, the programming task in ADTs includes the community code-formatting task.

Refactoring refers to practices that remove redundancy, eliminate unused functionality, and refresh obsolete designs (Fowler, 1999). Refactoring as an agile method reflects the team's commitment to improve the structure and reduce the complexity of the code whenever necessary (Beck, 2000). The entire ADT (rather than a particular subgroup, such as the architects) performs the refactoring whenever necessary. As such, refactoring combines software-development tasks that traditional practices might fragment.

2.3.4 Pair Programming and Combining Tasks

Pair programming refers to the practice of two developers working together to develop code (Cockburn & Williams, 2001). When pair programming, two developers work together to first design the software and then build it (Beck, 2000). In some cases, writing the tests is a light implementation of a design specification (Highsmith, 2002). When such an implementation occurs, some advocate that the first programmer write the tests for the code and the second provide feedback (Beck, 2000). Then, when they are programming the actual system code, the two developers reverse roles. Adding the development of the design to the programmers' responsibilities combines tasks that might be fragmented across analysts, architects, and developers in traditional team structures.

2.4 Forming Natural Work Units

2.4.1 Task Significance and Task Identity in Natural Work Units

Work fractionalization means that a person can complete only a portion of the work required to complete a full task, which can lead to workers addressing seemingly unrelated and unnatural sets of tasks in a given period (Hackman & Oldham, 1980). The lack of understanding of how work units are related leads to lower task identity and task significance. In contrast, by forming natural work units, employees develop ownership of their work (Hackman & Oldham, 1980). Developing this ownership mentality can increase the perceived meaningfulness and perceived value of the work. As such, forming natural work units increases task significance and task identity.

In traditional software-development environments, work fractionalization among various roles leads to assignments by task type, such as database coding, user interface coding, or testing. The contrasting approach used in ADTs is to assign as a bundle all of the tasks necessary to complete a particular feature rather than simply assigning a coder the next task in that coder's specialized area (Anderson, 2004; Palmer & Felsing, 2002). This practice is consistent with the call in the agile manifesto to define the key measure of progress in a development project as finished, functional code (Fowler & Highsmith, 2001). Rather than identifying their work as "data access layer coding", programmers in this example can identify their work as completing "the XYZ feature", which is an identifiable and functional component of the finished system—an accomplishment of definable significance. We propose that practices that allow coders to iteratively deliver code are more likely than traditional practices to lead to coders' forming natural work units and, ultimately, more clearly understanding task identity and task significance.

2.4.2 Iterative Delivery and the Formation of Natural Work Units

Although research has often discussed release planning, iteration planning, and velocity separately, they all concern iterative delivery (see Table 2). Release planning defines, at a high level, the order in which one will deploy features. Toward this end, ADTs and the customer plan iteratively before each work cycle; that is, the ADT's and customer's together define the features included in the next cycle, divide the features into tasks, and estimate the work they need to perform. These two practices also incorporate velocity. Velocity (as an agile practice) refers to calculating the amount of work that the ADT can achieve during a standard work cycle. As such, the calculated velocity of the ADT helps it to properly size the work committed to for the iteration. The agile experts consulted for this study (see Appendix A for details) repeatedly described these three concepts (i.e., release planning, iteration planning, and velocity) interchangeably, so we treat them as a single method called iterative delivery.

Iterative delivery helps ADTs to form natural work units. When the client provides priorities related to needed functionality, the team determines the work required to complete a full feature. Rather than focusing on developing particular modules of code, the team focuses on developing features, which are, by definition, natural work units.

2.5 Establishing Client Relationships

In the traditional software-development context, work is fractured and specialized and there is potential division between the analyst, architect, coder, and tester roles, which often creates several layers of hierarchy between an employee and the actual client. In this context, workers may not have contact with the people for whom the work has been completed. If employees lack direct contact with the customer, feedback is filtered through others, and employees lose a potential source of information through which they can understand the impacts of their work, which reduces task significance.

In contrast, agile practices emphasize the importance of direct and regular contact (in many cases on a daily basis) between the client and all the members of the team. This contact helps to establish client relationships. Direct client feedback obtained through these relationships is likely to provide the clearest information to help employees evaluate the overall success of their work.

Further, when employees establish relationships with their clients, employees gain increased skill variety and job autonomy. In any work structure, employees' technical or operational skills can grow through their work performance and training; however, when employees communicate with clients, employees can increase their skill variety by developing and using interpersonal skills. Additionally, when employees are empowered to directly manage a personal client relationship, their perceptions of job autonomy improve. Two agile project-management practices help to develop these client relationships: the daily stand-up meeting and iterative delivery.

2.5.1 Daily Stand-up Meeting and Establishing Client Relationships

The daily stand-up meeting is (usually) a daily meeting of the entire ADT that occurs while everyone stands to encourage brevity. Depending on the agile method in use, the daily stand-up meeting focuses on a short defined list of questions for each team member to answer (see Table 3). The ADT as a whole performs this practice each day. Each member of the ADT provides information regarding their work performed the previous day, their work planned for the day, and any blocking or coordination issues (Schwaber & Beedle, 2002). Agile methods also prescribe that, whenever possible, the business owner should attend this daily meeting.

By providing the opportunity to discuss the project's status, the daily stand-up allows each ADT member to directly interact with the customer. Each team member can ask the customer questions, which opens the door for a daily iterative dialogue and assists in establishing a direct relationship between the coding developer and the client. Further, because the daily stand-up focuses on issues (see Table 3), ADT members can be transparent with the customer (business owner), which builds trust, an important component of good relationships (Deutsch, 1960; Lewicki & Bunker, 1995; Lewis & Weigert, 1985).

2.5.2 Iterative Delivery and Establishing Client Relationships

Iterative delivery establishes client relationships because it creates an environment in which the customer and ADT share authority over the planning process. Both the client representatives and ADT members participate together in release planning and iteration planning. According to both scrum and XP, the client sets the task priorities, while the ADT estimates the work and establishes the schedule to complete those priorities.

Further, the ADT uses its defined velocity (Beck, 2000) to establish the amount of work that it can fit into a work cycle. By using this metric, which is determined based on the amount of work accomplished in previous cycles (see Table 3), the ADT sets an attainable goal. By completing iterations and releases as promised, the ADT as a whole establishes a trust relationship with the client, and each member of the ADT benefits from this relationship during regular interactions with the client.

2.6 Vertical Loading

Many traditional development contexts split software-development responsibilities between the planning (analysis and design), the doing (development and testing), and the controlling (project management) of the work, which creates a gap between each component of the job (Nerur et al., 2005). Vertically loading jobs reduce the distance between the doing, the planning, and the controlling of a given piece of work because vertical loading adds the “responsibilities and controls that formerly were reserved for higher levels of management” to individual jobs (Hackman & Oldham, 1980, p. 64). Vertical loading increases perceptions of job autonomy and personal responsibility because it returns discretion to the jobholder to make decisions regarding work scheduling, time management, and, to a lesser extent, troubleshooting, crisis management, and financial controls (Cordery, Morrison, Wright, & Wall, 2010; Hackman, Brousseau, & Weiss, 1976).

In a traditional software-development environment, one group of employees (e.g., architects, analysts) might make decisions about designing and implementing the system , while another set of employees (e.g., developers, testers) might complete the doing (or implementation), which indicates a lack of vertical loading. In contrast, ADT practices tend to vertically load the job by combining design, development, testing, and other tasks into a single team member’s responsibilities. We propose that a large number of the agile practices increase vertical loading, which we describe in Sections 2.6.1 to 2.6.3.

2.6.1 Iterative Delivery and Vertical Loading

To plan iterative delivery, the ADT must estimate the size of the work it needs to complete. When ADT members participate in planning and estimating, their roles are vertically loaded. ADT transforms software development by shifting control over planning and estimating from management or others to ADT members. By shifting control, ADT introduces more vertical control into team members’ jobs.

2.6.2 Retrospectives and Vertical Loading

Retrospective meetings occur at the end of a work cycle. At these meetings, ADT members reflect on the work process they used in the previous iteration and propose or adopt modifications to the process for the next work cycle. This process allows the ADT to direct itself, and, as such, the process vertically loads the ADT with the responsibility of defining its own development process (McHugh & Lang, 2011; Schwaber & Beedle, 2002). This process also gives each member a voice in evaluating work quality for the entire ADT (Beck, 2000; Highsmith, 2002; McHugh & Lang, 2011; Schwaber & Beedle, 2002).

2.6.3 Refactoring and Vertical Loading

Refactoring involves making changes to the internal structure of a software system to make the system code easier to understand and less expensive to maintain and modify (Fowler, 1999). Rather than relying on a coding supervisor such as a team lead or architect to complete this work, the agile software-development approach (SDA) practice of refactoring relies on vertically loading this important practice onto each developer in the ADT; refactoring in ADTs is a community activity. Rather than proposing to an architect or other leader that that they should change existing code, an ADT developer can refactor existing code. Agile methods encourage the use of this practice in order to encourage team members to feel that they have the job autonomy to ensure that the software system has as few defects and architectural issues as possible.

2.7 Opening Feedback Channels

Employees need to obtain additional information relevant to planning and doing their work and their work's results. Because feedback is one of the job characteristics that helps improve job satisfaction (see Table 1), additional opportunities and avenues through which to obtain feedback should increase this perception. Personal relationships and communications provide feedback (Hackman & Oldham, 1980); in addition, agile practices allow one to design feedback into the work itself. For example, when an employee's job of making a product or delivering a service includes quality-control task, the employee can obtain direct feedback and additional information about their performance. Agile development practices specifically merge design, development, and testing into a single process, which the same programmer or pair usually completes. These development processes, along with supporting automated tools, can immediately check the quality of a set of code and provide feedback directly to the individual who created it.

2.7.1 Daily Stand-Up Meeting and Opening Feedback Channels

The daily stand-up meeting opens multiple feedback channels by allowing ADT members to immediately judge the quality of their previous and current work processes and outputs. When ADT members discuss their status daily with all members of the team, they can share feedback with one another about completed work, work barriers, and issues that previously performed work caused. These repeated, planned discussions allow the team to provide and receive feedback and to understand the status of the project's environment (Burke, Stagl, Salas, Pierce, & Kendall, 2006). Further, because the client attends the meeting as well, each member of the ADT can receive direct feedback about questions regarding direction, requirements, or issues.

2.7.2 Retrospectives and Opening of Feedback Channels

Retrospective meetings occur at the end of a work cycle and provide a feedback channel for members in the ADT to evaluate their work quality (Beck, 2000; Highsmith, 2002; McHugh & Lang, 2011; Schwaber & Beedle, 2002). Retrospective meetings allow ADTs to continuously improve because they provide a forum in which each member can describe what is working well, what is not working well, and how things need to be changed in the next work cycle (Derby, Larsen, & Schwaber, 2006). Retrospectives allow team members to clarify issues from previous iterations because these meetings provide clear feedback about whether and how the team achieved previous work cycles' goals and allow team members to define opportunities to improve work practices for the next work cycle (McHugh et al., 2012). Research has found the retrospective's omission from ADT processes to contribute to a lack of shared understanding in the team about whether it is performing well (Moe et al., 2010).

2.7.3 Burndown and Opening of Feedback Channels

The burndown chart is a feedback channel that compares the amount of work planned at a given point in a work cycle with the amount of work actually completed (Sutherland, 2001). The entire ADT (rather than just a project manager) sees this chart, which provides all team members with the necessary information about the progress they have made toward the previously set team goal. By providing the ADT with feedback about its current progress toward a goal, the burndown chart empowers each member of the ADT to take action to reduce the possibility of the project's falling behind.

2.7.4 Automated (Unit) Testing, Automated Builds, and Opening Feedback Channels

These three practices work together to open feedback channels for each individual developer and for the team as a whole. First, automated unit testing opens feedback channels by providing immediate information to a developer via a failed test. Many development teams require coders to run the full suite of tests before committing code changes to the common code repository (Beck, 2003; Highsmith, 2002; Küster et al., 2009). By iteratively running all of the tests until they pass, ADT members obtain information that allows them to immediately validate the quality of the code they have written.

Automated builds extend the practice of automated (unit) testing. Automated builds allow any developer to rebuild the entire software system after refreshing from the repository or at any other time before committing. Automated building goes beyond automated unit testing because the practice of performing automated builds ensures not only that the system code is functional but also that the team has added all other build dependencies to the list of system dependencies (for instance, a third party code component) and to the build scripts. This process provides a feedback channel to each individual who is changing the system so that the team can build their changes into the updated software system without issues.

2.7.5 Continuous Integration and Opening Feedback Channels

Continuous integration (CI) extends to the entire ADT the feedback channels opened to individuals through automated testing and automated builds. Because continuous integration occurs on a server machine that is common to the entire team and refreshed after each team member checks in code, it allows the ADT as a whole to ensure that the code is consistently deployable. This opens a feedback channel that allows the entire ADT to determine whether members have completed a change properly. CI provides feedback beyond the process of automated testing and automated builds, because it ensures not only that the code runs on a developer's machine but also that the developer has committed all of the necessary configuration changes to the code repository (Duvall et al., 2007; Fowler, 2006).

2.7.6 Pair Programming and Opening Feedback Channels

Working in pairs better enables coders to identify mistakes during the actual coding process. When the observing coder notices a mistake in progress, that person can give immediate feedback to correct it. Further, because agile methods argue that pairs should be fluid structures, they often recombine regularly (in some cases, pairs can change more than once per day) (Beck, 2000); in this way, feedback channels are opened between the pair to ensure that the entire ADT consistently implements coding standards.

In summary, this section defines each of the five job-design principles and explains how each of the 10 agile practices relates to each principle. We suggest that job characteristic perceptions increase when ADT practices that instantiate the job-design principles design and recombine jobs. In Section 2.8, we map agile practices to the job-design principles and develop a model that connects agile practices, job characteristics, and job satisfaction.

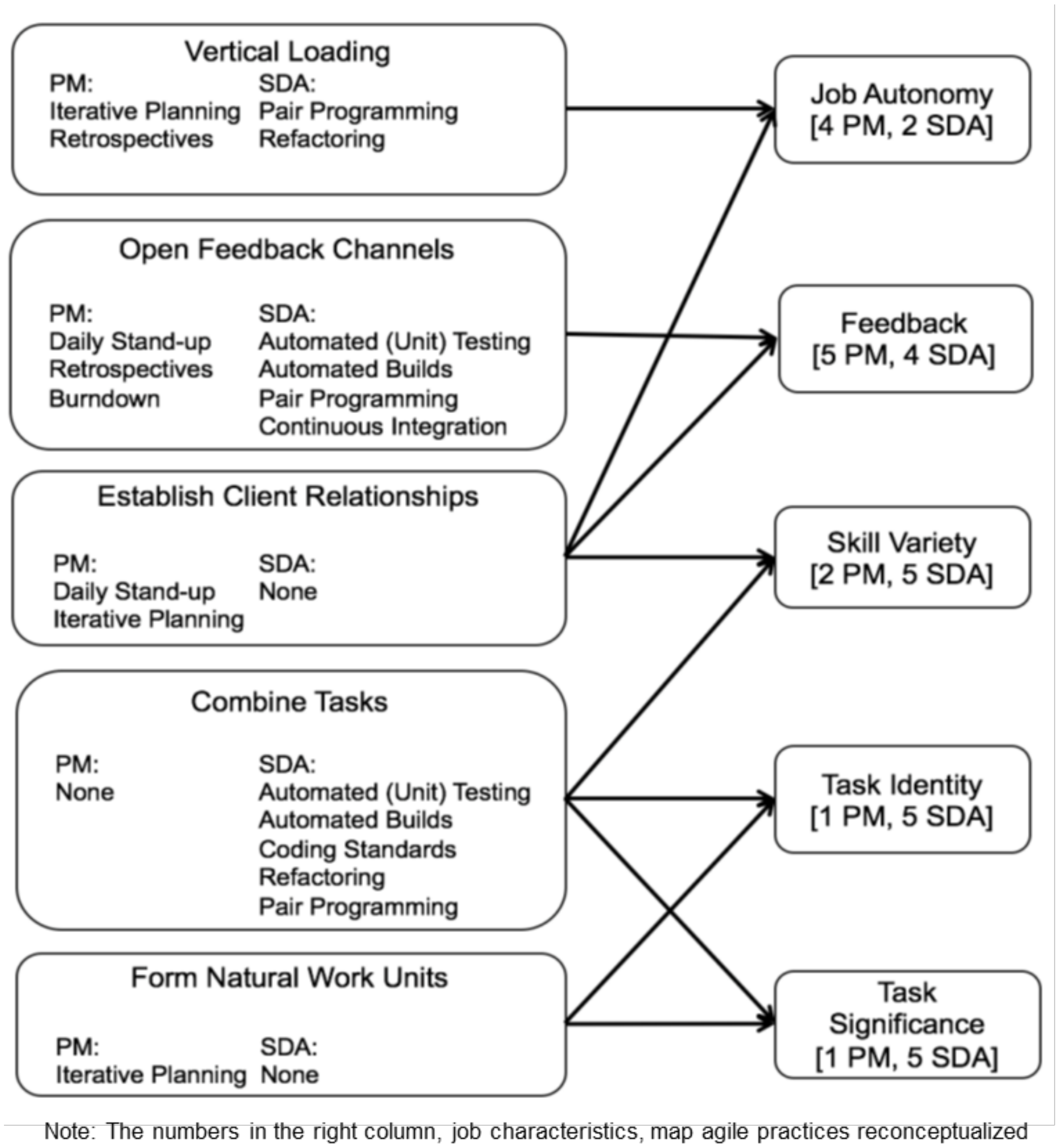
2.8 Operationalizing and Mapping of Agile Practices to Job-design Principles and the Job Characteristics Model

Table 2 categorizes the 10 agile practices into two groups: agile project management (PM) practices and agile software-development approach (SDA) practices. We believe that one can best conceptualize these groupings as distinct, second-order multidimensional constructs. They are distinct because agile PM practices and agile SDA practices have different foci and different impacts on perception of job characteristics. Agile SDA practices focus primarily on executing the software-development process. In contrast, agile PM practices primarily focus on establishing customer relationships, obtaining feedback, and coordinating work in teams. As such, the agile PM practices are primarily managerial and support the performance of the actual software engineering. Because of these different foci, one can represent agile practices as two conceptually related yet distinct constructs.

We believe one needs to model each of the 10 individual agile practices as reflective in nature (Petter, Straub, & Rai, 2007; Polites, Roberts, & Thatcher, 2012). Because terminology is inconsistent across methods and because one team might claim to use a practice but not actually implement it in the same manner as another team, it is difficult to evaluate how much teams use each practice. To that end, in our survey (described in Section 4) we asked multiple agile practice-specific questions; for example, the use of stand-up meetings and iterative delivery help to determine the depth to which an ADT employs agile PM practices. For these first order factors, we asked questions that sampled the activity domain of a team's practices to evaluate the depth of each agile method's use. Thus, to evaluate the extent of stand-up meeting use, for example, we asked questions such as the frequency and range of issues discussed at meetings. Although these questions tap into the broader domain of stand-up meetings, one could conceive them as interchangeable.

We conceptualized the second-order factors as formative in nature (Polites et al., 2012) because the use of one agile practice (e.g., pair programming) might not require the concurrent use of other agile practices. Because of most agile practices' independence, using one practice at a high level does not necessarily mean that a team uses another practice at a high level. Our agile-practices conceptualization suggests that, to effectively evaluate whether teams use agile PM and agile SDA, one must evaluate not only the depth of a practice's use but also whether the team employs other agile practices. Hence, while at the first order we used multiple interchangeable items to evaluate a practice's use (e.g., in a reflective manner), our second-order conceptualization required that we also aggregate the presence of specific practices employed by an ADT. For this reason, at the second order, agile PM and agile SDA practices should be operationalized as formative.

As our review and associated measurement theory suggests, one can conceptualize PM and SDA practices as aggregate constructs; that is, one can conceptualize them as having reflective first-order dimensions and a formative relationship from the first order to the associated second-order construct. In Figure 2, we conceptualize the agile practices as acting as instantiations of job-design principles. Our explanation of the figure further underscores the importance of modeling practices as distinct constructs at a second-order level. The agile PM and the agile SDA practices highly overlap on the vertical loading and opening-feedback channels concepts: more than 50 percent of each category's practices instantiate these two job-design principles. In contrast, only agile PM practices instantiate establishing client relationships and forming natural work units, while only agile SDA practices instantiate combining tasks.



Note: The numbers in the right column, job characteristics, map agile practices reconceptualized as instances of the job design principles, using Hackman & Oldham's (1980) job characteristics model (JCM). For example, because two job design principles constructs (left column) lead to job autonomy, the instances from both vertical loading and establishing client relationships are added to generate the numbers in the right column. Because some practices are instantiated in multiple ways, the impact of the same agile practice can be counted multiple times.

Figure 2. Agile Practices as Instances of Job-design Principles Mapped to JCM

Extending the mapping, the right column of Figure 2 lists the five job characteristics and the cumulative instantiations of the job-design principles through the use of each agile practice. To illustrate, job autonomy category has four agile PM practices (two from vertical loading and two from establishing client relationships) and two agile SDA practices (two from vertical loading and zero from establishing customer relationships). As the figure shows, the job autonomy and feedback categories each have a relatively high number of the agile PM and agile SDA practices mapped to them, while the remaining job characteristics are primarily mapped to agile SDA practices. Hackman and Oldham (1980) argue that the three constructs of task identity, skill variety, and task significance are primarily related to job tasks, while the constructs of feedback and job autonomy are related to job management. Similarly, the agile SDA construct comprises practices that primarily focus on the actual process of performing software development, while the agile PM practices focus on job management (Tripp & Armstrong, 2014). The implications of these differences in mapping directly affect our research model, and we discuss them in Section 3.

In summary, in this section, we describe Hackman and Oldham's (1980) job characteristics model and job-design principles. We also illustrate numerous ways in which the one can conceive the practices defined in agile methods as instantiating the job-design principles. We conceptualize PM and SDA practices as second-order, aggregate constructs. Finally, we present a mapping of agile practices to job-design principles and then to the job-characteristics model. In Section 3, we present our research model and hypotheses.

3 Hypothesis Development

Figure 3 presents our research model, and Table 1 (see Section 2.1) defines our constructs.

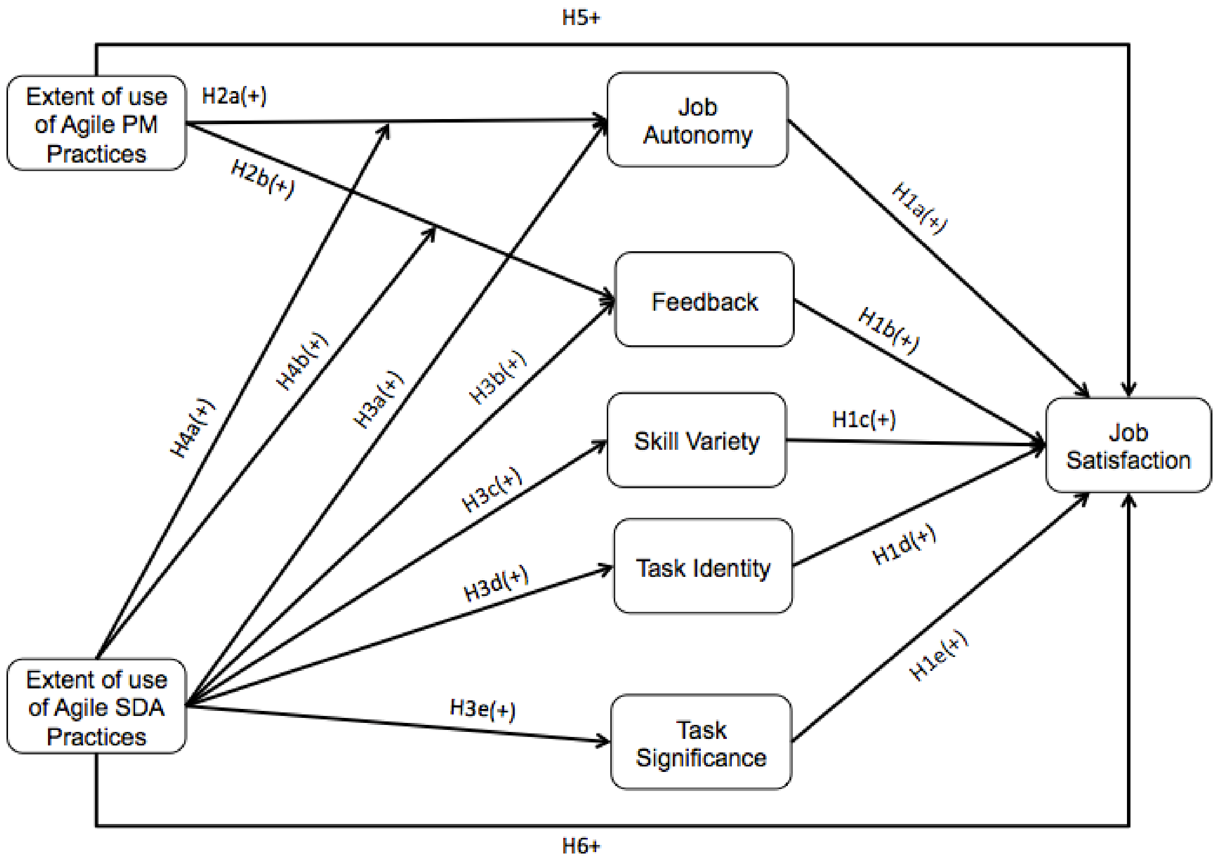


Figure 3. Research Model

Our research model leverages the logic articulated in prior sections of this paper and extant research. Research has extensively tested our first set of hypotheses (e.g., Ahuja et al., 2007; Morris & Venkatesh, 2010; Thatcher et al., 2002). We include these hypotheses as a means to connect our model to the established IT workforce nomological network (Joseph et al., 2007). Hence, we hypothesize that employees' more favorable perceptions of job characteristics results in more favorable perceptions of job satisfaction. As such, we hypothesize:

H1a-e: Higher perceptions of the job characteristics of a) feedback, b) skill variety, c) task identity, d) task significance, and e) job autonomy are positively related to job satisfaction.

Our review of agile development methods and job design principles suggests that when agile practices use increases, ADT members should see their jobs as less fragmented, more autonomous, and more significant. We argue that agile PM and agile SDA practices are related to job characteristic constructs in different ways. First, we hypothesize that the extent of agile PM practice use primarily impacts job characteristics that are related to managing the job rather than executing the software development task. As Hackman and Oldham (1980) argue, the job characteristics feedback and job autonomy are primarily related to job management. Using agile PM practices increases vertical loading, opens feedback channels, and allows the entire team to engage with the client. These forces are reflected primarily in feedback and job autonomy, two important job characteristics. Therefore, using these job-design principles should increase positive perceptions of feedback and job autonomy. As such, we hypothesize:

H2a-b: The extent of use of agile PM practices positively impacts the perceived job characteristics of a) job autonomy and b) feedback.

In contrast to the agile PM practices, which focus on managing a project, the agile SDA practices focus on completing the actual task. As such, agile SDA practices will likely impact the task-related job characteristics: task significance, task identity, and skill variety. This occurs primarily through the agile practices that combine tasks that are fragmented in traditional work models (e.g., developer tasks are combined when a developer is allowed to write tests and production code components for various features rather than handing testing over to another person).

Further, consistent with the mapping in Figure 2, we argue that agile SDA practices also have the potential to impact job autonomy and feedback. Agile SDA practices open feedback channels in several ways. First, automated testing and continuous integration allow ADT members to quickly evaluate the impact of any code change. By its nature, pair programming provides feedback to the coder because the partner continuous evaluates the work as it occurs. Finally, agile SDA practices provide numerous opportunities to vertically load a job, which can lead to more opportunities to be autonomous.

Because one can map the agile SDA practices to all five job characteristics, we hypothesize:

H3a-e: The extent of use of agile SDA practices positively impacts job characteristic perceptions of a) job autonomy, b) feedback, c) skill variety, d) task identity, and e) task significance.

Our mapping of agile practices in Figure 2 suggests overlap in the potential impacts of both agile PM practices and agile SDA practices on both job autonomy and feedback. The impact of using both agile PM and agile SDA practices would likely positively reinforce the impacts of each construct. Although not yet tested, agile practitioners have long argued that the impact of using multiple practices is not simply additive but multiplicative (e.g., Beck, 2000; Highsmith, 2002); that is, that multiple practices yield an impact greater than the sum of individual parts. As an example, automated testing and continuous integration likely increase iterative delivery's impact, which allows the ADT to make fast course corrections and direction changes as necessary. As such, we hypothesize:

H4a-b: There is a positive interaction effect between the extent of agile PM practices use and the extent of agile SDA practices use on a) job autonomy and b) feedback.

Additionally, we propose that agile practices have direct effects on job satisfaction. Agile proponents report and some research has supported that ADTs have higher engagement with the client and higher developer motivation (e.g., McHugh et al., 2010). Research has also found ADTs to deliver higher-quality software (Maruping et al., 2009a) and have shorter project timelines and higher project success rates than traditional team methods (West et al., 2010). Agile teams work iteratively and set short-term goals to achieve working and deliverable software every few weeks (Highsmith, 2002). Research has viewed job satisfaction as a product of rewards (tangible and intangible) gained from performance (e.g., Naylor, Pritchard, & Ilgen, 1980; Vroom, 1964). Agile team members can gauge their individual performance daily by using feedback practices such as unit testing, continuous integration, and burndown. Further, ADTs gauge their performance at the end of each work cycle and receive feedback from their customers about the quality and suitability of their work (Beck, 2000; McHugh et al., 2010; Schwaber & Beedle, 2002). This shortening of the performance-evaluation cycle should lead to greater perceptions of satisfaction because it affords opportunities for recognizing success and improving continuously (Burke et al., 2006). As such, we hypothesize:

H5: The extent of use of agile PM practices positively impacts job satisfaction.

H6: The extent of use of agile SDA practices positively impacts job satisfaction.

In Section 4, we present the details of our research study that tested these hypotheses.

4 Research Method

To evaluate our research model, we collected survey data in June 2013 from 252 respondents who worked as software-development professionals. We recruited our sample using Empanel, a data-collection company that specializes in recruiting Internet-based survey panels. The panel comprised IT professionals screened by Empanel. Many IS studies have successfully collected and use data in this manner (e.g., Angst & Agarwal, 2009; Ayyagari, Grover, & Purvis, 2011; Benlian, Titah, & Hess, 2012). The company provided 1,098 potential respondents. We asked each potential respondent whether they were currently part of a software development team and what role they played on the team. Further, to ensure that our respondents had sufficient background and experience to have developed perceptions of their current jobs, we asked them to report their total IT experience and how much experience they had at their current organization[3]. Based on the answers to these screening questions, we eliminated 441 potential respondents and included the other 657 respondents in the survey.

As additional quality checks, we positioned "quality assurance" questions at several points in the survey to ensure that respondents were engaged with it. If respondents answered a QA question improperly, we immediately terminated their survey and dropped their responses. As a result, we disqualified 351 respondents who had started the survey. Our questionnaire was long, so, to ensure our analysis included only respondents who carefully thought about each question, we rejected 51 respondents who completed the questionnaire in less than 10 minutes[4]. This left 255 respondents who completed the full survey, who answered all of the QA questions appropriately, and who took at least 10 minutes to complete the survey. Finally, we reviewed these responses and dropped three additional respondents who answered all of the questions the same or indicated in an open-ended question that they did not qualify. We included data from the remaining 252 respondents in the study. Of the 252 respondents, 125 indicated that they worked in agile development teams, and 127 indicated that they did not. Of the 252 respondents, 80.7 percent were male. This percentage is consistent with reported gender ratio analyses of software development teams (James, 2010). Appendix A2 breakdowns the sample by team role.

Where possible, the survey included scales adapted from previous research. To measure agile method use, we developed a new set of items. Table A1 presents the sources for the items and the instrument we used for this survey.

5 Model Analysis

We analyzed our theoretical model using partial least squares (PLS) with SmartPLS (Ringle, Wende, & Will, 2005). Partial least squares structural equation modeling (PLS-SEM) is an appropriate analysis technique for our model because the model is relatively complex, includes interaction effects, and models agile method practices as aggregate second-order constructs (Polites et al., 2012). We performed multiple tests to validate our model, including tests for convergent and discriminant validity and common method bias, the results of which Appendix B reports. Our results suggest that our model meets or exceeds standards for validity (Straub, Boudreau, & Gefen, 2004).

We modeled agile PM and agile SDA constructs as aggregate constructs, where the agile practices were measured reflectively, and then the second-order factors were composed of the respective practices (Hair, Hult, Ringle, & Sarstedt, 2013). We used a multi-stage approach for the analysis; we modeled the approach after hierarchical regression based on the technique that Siponen and Vance (2010) applied. First, we examined the independent effects of the control variables (negative affectivity, age, organizational tenure, gender, education, and total work experience—see Appendix A) without including any of the JCM or agile practice constructs on the dependent variable job satisfaction. We call this model 1. These variables

[3] Respondents had to play a role as a member of a software development team, have at least one year of total experience, and have worked for their current organization for at least six months.
[4] We later analyzed our data including these 51 respondents. Our results did not substantially change.

explained 20 percent of the variance of job satisfaction. However, inspecting the analysis suggested that only negative affectivity and gender significantly related to the dependent variable.

Next, we analyzed the added effects of the job characteristics model's constructs (feedback, job autonomy, task identity, task significance, and skill variety), which we refer to as model 2. Of these constructs, all were significant (all $p < .05$). The variance explained in model 2 increased from 20 percent to 54.2 percent. We then calculated the size of the effect differences between the two models as $f^2 = (R^2_{Model2} – R^2_{Model1})/(1 – R^2_{Model2})$ (Chin, Marcolin, & Newsted, 2003; Qureshi & Compeau, 2009) in order to test whether this increase was significant. Then, we performed a pseudo F-test by multiplying the effect size (f^2) by ($n – k – 1$), where n is the sample size and k is the number of independent variables (Mathieson, Peacock, & Chin, 2001). The size of the effect was very large (.741), indicating that the change in R^2 was significant ($F = 181.04$, $p < .01$), which means that the job characteristics model construct's impact on variance explained in job satisfaction (model 2) was significantly higher than that explained by model 1.

Table 4. Model Comparison

	Model 1 controls only	**Model 2 JCM + controls**	**Model 3 full model**
R2	.200	.542	.582
Delta R2		.342**	.040*
Effect size		.747	.096
**p <.01, *p<.05 JCM job characteristics model			

Finally, we added the agile practice use constructs to complete the full model (model 3) (see Figure 2). The full model increased the explained variance in job satisfaction from .542 to .582. The effect size was .096, which is a small effect (Cohen, 1988). However, even though the effect was small and the change in explained variance was only 4 percent, this change was significant when we performed the pseudo F-test ($F = 14.55$, $p < .05$). These tests support including the agile practice use constructs in the full model. Table 4 summarizes this multi-stage analysis. To test significance, we performed bootstrapping with 252 cases and 1,000 iterations; Figure 4 presents the results of the full model.

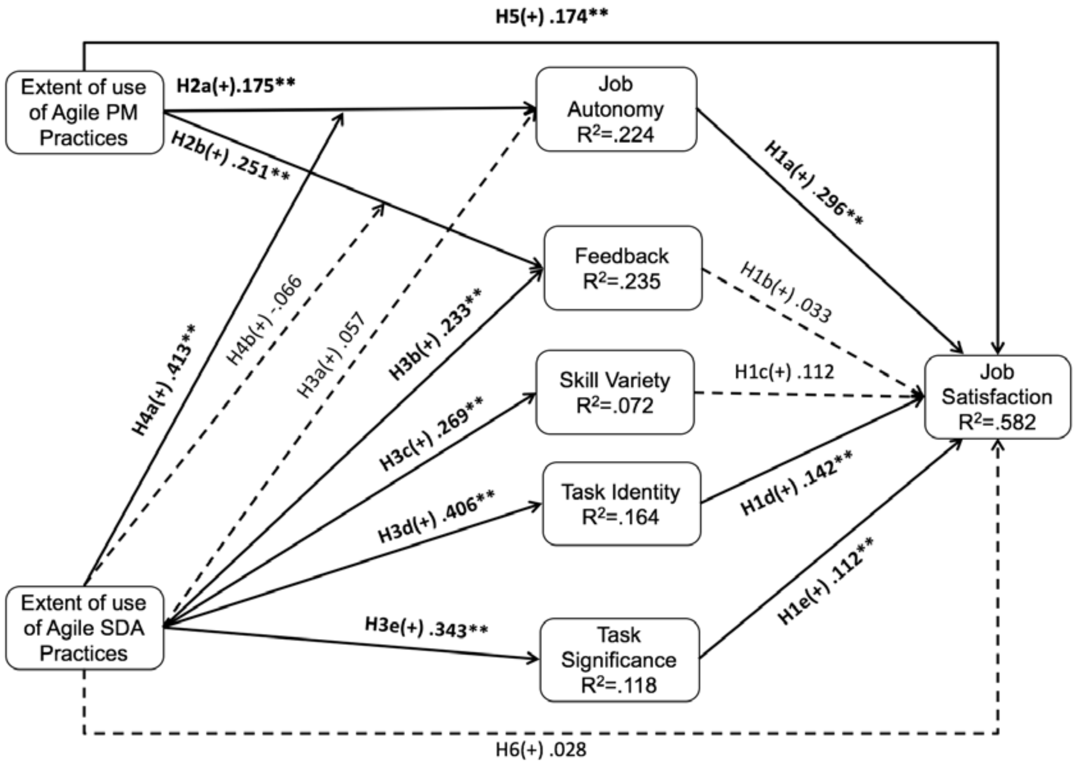


Figure 4. PLS Results

Interestingly, while all paths from the job characteristics to job satisfaction were significant in model 2, in the full model, the paths from feedback (.033) and skill variety (.112) were not significant. Also, although we hypothesized that both agile practice use constructs have a direct effect on job satisfaction, only the agile PM practices (.174) were significant at the p <.01 level. All of our hypothesized paths from the agile practice use constructs were significant except the path from agile SDA practices to job autonomy. The interaction between agile SDA practices and agile PM practices significantly influenced job autonomy and did not influence feedback. Thus, our results partially support H1, H3, and H4 and support H2 and H5; they do not support H6. Table 5 summarizes our hypothesis results. Also, in our final model, negative affectivity was the only control variable that significantly related to job satisfaction. In Section 6, we discuss our findings and their implications.

Table 5. Summary of Hypothesis Results

Hypothesis	Result	Supported?
H1: Higher perceptions of the job characteristics are positively related to job satisfaction. a) Job autonomy b) Feedback c) Skill variety d) Task identity e) Task significance	 .296** .033 .112 .142** .112**	 Yes No No Yes Yes
H2: The extent of use of agile PM practices positively impacts the perceived job characteristics of a) job autonomy and b) feedback a) Job autonomy b) Feedback	 .174** .251**	 Yes Yes
H3: The extent of use of agile SDA practices positively impacts job characteristic perceptions of: a) Job autonomy b) Feedback c) Skill variety d) Task identity e) Task significance	 .057 .233** .269** .406** .343**	 No Yes Yes Yes Yes
H4: There is a positive interaction effect between the extent of agile PM practices use and the extent of agile SDA practices use on: a) Job autonomy b) Feedback.	 .413** -.066	 Yes No
H5: The extent of use of agile PM practices positively impacts job satisfaction.	.174**	Yes
H6: The extent of use of agile SDA practices positively impacts job satisfaction .	.028	No
Note: ** - p < .01		

6 Discussion and Implications

Motivated by a desire to evaluate practice-based claims about agile practices' positive influence on job design, we investigated two research questions:

RQ1: How does the extent of use of different agile practices impact job satisfaction?

RQ2: What are the causal pathways through which agile practices affect job satisfaction?

Through a cross-sectional investigation of ADT members, we found support for agile practices' affecting job satisfaction. First, we found that members of ADTs that used higher levels of agile PM practices had higher perceptions of job satisfaction (Hypothesis 5). Conversely, higher levels of the use of agile SDA practices did not have a direct impact on perceptions of job satisfaction (Hypothesis 6). Second, we found that the extent of use of agile practice constructs impacted perceptions of job characteristics, although not all of our predicted paths were significant (Hypothesis 3a). We also found a positive interaction between agile PM practices use and agile SDA practices use on job autonomy but not on feedback. We discuss each of these findings below.

While we found general support for agile practices' affecting job satisfaction, we found that not all job characteristics significantly influenced job satisfaction in the presence of agile method use. In model 2, which did not include the agile method use constructs, all five job characteristics impacted job satisfaction significantly as one would expect based on extant research (e.g., Ahuja et al., 2007; Hackman et al., 1976; Moore, 2000; Morris & Venkatesh, 2010; Ply et al., 2012). However, in the full model (model 3), which considers agile method use, only three of the job characteristics (job autonomy, task identity, and task significance) were significant. We found that feedback and skill variety were insignificant in the full model. A possible explanation for these results comes from the inherent benefits embedded in the design of agile practices. Two of the benefits of implementing agile practices are the high occurrence of iterative feedback and the increased skill variety. It is plausible that the measures focused on feedback in agile practices were more salient to the respondents than the more general measures typically used to measure feedback in the JCM. Further, as theorized, the same effect might occur for the skill variety construct because both agile PM and agile SDA practices require using a more diverse skill set than might be required in a typical traditional software-engineering environment. We believe that, while job characteristics are an excellent predictor of job satisfaction, research should develop additional theory regarding the impact of various IS-related contextual factors on the formation of perceptions of the job characteristics. This study represents a first step toward developing contingency models that explain how the structure of IT work makes job characteristics more or less salient for ADT members and the broader IT workforce.

By examining sources of job characteristics, our work advances the IT workforce literature beyond its foundation in the JCM. In our analysis, the impact of the extent of use of agile PM practices was highly significant with regard to feedback and skill variety, which supports Hypotheses 2a-b. As expected, the agile SDA practices were associated with higher perceptions of the task-specific job characteristics, including task identity, task significance, and skill variety. The extent of use of agile SDA practices impacted feedback as well, although the path to job autonomy was insignificant. Job autonomy might not be salient to agile teams because agile SDA practices reinforce the interdependence of team members' work. For example, agile practices such as pair programming and coding standards focus direct attention on individuals' responsibility for adhering to team norms and agreed-on processes. Despite this non-finding, these results confirm proponents' assertion that using agile methods improves perceptions of job characteristics (McHugh & Lang, 2011; Tessem & Maurer, 2007). Hence, this paper importantly demonstrates that software-development practices such as agile methods relate to how ADT members perceive their jobs.

Moreover, our study suggests that sets of practices shape core job characteristics. We found that agile PM and SDA practices interacted to positively influence job autonomy (Hypothesis 4a). The interaction effect on job autonomy reflects that the ability to manage and change direction iteratively (as reflected in the agile PM practices) is likely contingent on using complementary software-development approach practices. Agile proponents argue that creating a full design up front is likely to waste resources because understanding of the business problem changes over time. However, this approach can lead to large sections of code having to be repeatedly refined and potentially added and removed. Without agile SDA practices such as unit testing and automated builds, the impact of large-scale changes would impact a team's ability to realize the iterative development benefits. Thus, our findings support agile proponents' long-argued assertion that agile practices, when used together, produce an impact that is greater than the sum of their parts (e.g., Beck, 2000; Highsmith, 2002).

We found that the interaction of the agile PM and agile SDA constructs did not consistently affect perceived job characteristics. Although the agile SDA and agile PM constructs had a direct effect on feedback, the interaction effect was non-significant (Hypothesis 4b). This result could be because the feedback practices in the two constructs are different. We mapped the agile PM practices to feedback through the JDP constructs of establishing client relationships and opening feedback channels, while we mapped the agile SDA practices through only opening feedback channels. ADT members may perceive feedback generated by the agile PM practices such as the stand-up meeting and the retrospective as mainly human-generated feedback. In contrast, they may perceive the feedback generated by the agile SDA practices such as automated testing and continuous integration as machine-generated feedback. Although the interaction of agile PM and agile SDA practices did not affect feedback, future research should explore whether the difference in these practices affects other workplace outcomes, such as professional rewards, or more frequently researched outcomes such as job satisfaction and ADT member turnover.

Our findings underscore that agile PM and SDA practices exert differential effects on how individuals perceive their workplace. Turning to Hypothesis 5, we found a direct effect between the level of agile PM practice use and job satisfaction. Agile PM practices focus on organizing to rapidly and repeatedly deliver software. Because regular and repeated response to environmental cues is critical to assessing a team's success (Burke et al., 2006), using agile PM practices will likely lead team members to view their work as successful and, therefore, satisfying. This finding is consistent with the prior literature that describes coordination practices as impacting job satisfaction (e.g., McHugh & Lang, 2011). Conversely, our results did not support Hypothesis 6 because the agile SDA practice use level did not affect perceptions of job satisfaction directly. Agile method proponents have argued that some practices support using other practices. Instead of having a direct effect, agile SDA practices might support a team's ability to deliver iteratively. If this were the case, then agile SDA practices would have an indirect effect on job satisfaction.

6.1 Tests for Mediation

Given our findings, we performed post-hoc analyses to explore whether our direct effects were partly (agile PM) or fully (agile SDA) mediated by perceptions of job characteristics. Using the seemingly unrelated regression (sureg) package in Stata 13, we tested for multiple mediation as Preacher and Hayes (2008) recommend. We tested each indirect path effect individually and then pooled the agile PM effects and agile SDA effects to test the total indirect effect of each construct. Because indirect effects are usually skewed and have high kurtosis, the p-values to test significance are not robust. We used bootstrapping (5,000 replications) to determine significance. Table 6 shows the results. We found no mediation effect of the agile PM construct and a small fully mediated effect (.091*) of the agile SDA construct. Therefore, the full effect of the use of agile PM practices on job satisfaction was direct, while the full effect of the use of agile SDA practices was fully mediated by the JCM constructs of job autonomy, task significance, and task identity.

Table 6. Multiple Mediation Effects

Agile PM -> job autonomy -> job satisfaction	-.003
Agile PM -> feedback -> job satisfaction	.000
Total agile PM mediation effect	-.003
Agile SDA -> job autonomy -> job satisfaction	.024*
Agile SDA -> feedback -> job satisfaction	.000
Agile SDA -> task significance - > job satisfaction	.043*
Agile SDA -> task identity - > job satisfaction	.034*
Agile SDA -> skill variety - > job satisfaction	-.010
Total agile SDA mediation effect	**.091***
Note: *p<.05	

One can possibly explain this difference in direct versus indirect effects for the agile PM and agile SDA constructs by the fact that agile SDA practices are more individually task related, while agile PM practices give team members a sense of control over a broad set of project characteristics. This distinction is important because the JCM describes individuals' control over their own individual tasks and context, while the agile PM practices give individuals a voice in a broader team- and project-level context.

6.2 Implications for Theory and Future Research

By connecting agile development practices to perceived job characteristics and satisfaction, our results hold the potential to inform future research. First, while research has investigated the impact of technology characteristics on job characteristics (e.g., Lending & Chervany, 2002), our study is the first to examine a specific IS development practice (e.g., using agile practices) as an antecedent to job characteristics. As such, our results signal a new avenue for researchers seeking to develop a theory of how to manage the IT workforce. Previous studies have often treated the job characteristics as exogenous and have generally investigated IT contexts using the model that Hackman and Oldham (1980) describe (e.g., Ahuja et al., 2007; Moore, 2000; Thatcher et al., 2002) or have used Hackman and Oldham's model as a lens to investigate changes in perceptions resulting from IT implementations (e.g., Morris & Venkatesh, 2010). However, our study shows that IS-related organizational and process constructs can impact the formation of the perceptions of job characteristics. Further research should explore the impact of other IS-related constructs on the formation of JCM construct perceptions.

Additionally, by distinguishing between agile PM and agile SDA practices and the differences in their effects, we illustrate the importance of taking a more nuanced view of software-development practices. This more nuanced view suggests that, rather than discussing “agile development methods” as a monolithic concept, future research needs to carefully consider the foci and structure offered by the method or practice relationship with job satisfaction and other perceptions. For instance, our results suggest that some agile practices affect the individual directly and that others affect the team as a whole. Additionally, our results hint at a more complex structure of agile practices’ influencing workplace outcomes, where some practices support others. Agile practitioners have argued for the compounding effect of using multiple agile practices and a structure of reinforcing practice use that enables other practices (Beck, 2000). Research has also identified similar non-linear effects of using multiple practices in the human resource-management context (Ferratt, Agarwal, Brown, & Moore, 2005). Therefore, developing an understanding of whether a nomological network of constructs is at work in the context of agile methods use would be a significant contribution. Such research is important because it could enable managers to properly match agile methods to projects such that they more positively affect both product quality and employee satisfaction.

Furthermore, our findings lay a foundation for research to explore agile PM and agile SDA practices as they relate to relevant job perceptions and behaviors such as perceived work overload, burnout, empowerment, performance, and turnover. Such research should consider not only the organizational impact but also the impact of agile practices versus traditional practices over time (e.g., the life of a project or a professional’s career). For example, as an ADT member gains experience across various projects and varied sets of agile practices, that ADT member will likely develop a richer understanding of the impact of agile practice use. Because most prior work treats agile development as a monolithic concept, research has relatively unexamined agile practice use perceptions, project success results, resulting job characteristics changes, and the impact of these factors on ADT members over time. Further, future research might explore the relationship between agile project success and job satisfaction because the effects of team performance and individual job satisfaction are not independent (e.g., Cox, 2003).

Beyond agile methods, researchers should evaluate the effects of project success on ADT members’ beliefs about the implications of ADT practices. For example, agile methods open feedback channels through acceptance tests that client representatives or product owners write[5] (Matook & Maruping, 2014). People in this role need to approve or disapprove the agile development team’s implementation of specific features, and the team receives feedback on how well its software is meeting client specifications. Yet, while research has found that such changes have positive effects on perceived job characteristics, we know little about how such changes affect ADT members’ global perceptions of their jobs, their projects, or the broader employing organization. It is intuitive that ADT members who perceive that agile practices use is related to higher project success would believe that using those practices is a positive event. Conversely, if ADT members believe their projects are not successful, they may find using agile practices as exhausting with a potential reduction in job satisfaction. Research should explore the relationship between agile PM and agile SDA practices and ADT employee perceptions of project success and job- and organizational-related outcomes such as burnout or commitment to the organization.

Although we conceptualized this study at the individual level, there are rich opportunities for multi-level research on software development and agile practices. Because of the highly interdependent nature of team-based software development work, research should study our theory should at the team level or with a multi-level approach across organizations. Such research could yield additional insight on the within-team and between-team differences in job satisfaction by including multiple respondents from the same team. For example, examining the interplay among industry, agile practices, and leadership practices could yield further insight into ADT members’ job satisfaction and other workplace perceptions.

Finally, we acknowledge that we need to move beyond the job characteristics model to fully understand the effect of agile PM and agile SDA practices on agile development team members (Behson, 2012). Agile teams have a great deal of input into the demands of their jobs (e.g., iterative planning, retrospectives). As such, Karasek’s (1979) job strain model, which argues that a balance between authority and responsibility leads to an employee’s being free to take action, might provide a new lens to investigate the impact of agile methods on team performance and job satisfaction. Given that ADTs are intentionally self-directing and often perform highly, the job-strain model might provide a lens to conceptualize a novel theoretical approach to job satisfaction in ADT. In addition, the JCM is only one small component of the research on job design and its outcomes (e.g., Grant, Fried, & Juillerat, 2011; Joseph et al., 2007). Future research on agile methods could

[5] Thanks to an anonymous reviewer for this suggestion.

investigate the impacts of the agile PM and agile SDA constructs on several additional outcomes such as learning, performance, and stress and interactions with other constructs such as interdependence. We still have much to learn regarding the impact of agile methods as catalysts of job redesign.

6.3 Implications for Practice

Our study has several implications for practice. First, it provides support for a key argument made by agile proponents (Beck, 2000) in that it shows that a broad approach of adopting both management- and task-related agile practices leads to non-linear impacts on the organization. This non-linear effect suggests that management should offer incentives and encourage teams to move beyond exploratory agile implementations and to implement more agile software-development practices. Even though industry surveys have found that agile practice adoption is widespread, many if not most such projects use relatively few agile practices (Conboy & Fitzgerald, 2010).

However, our research also shows that more does not necessarily mean better at least when it comes to the impact of agile PM and agile SDA practices on job satisfaction. Our results suggest that the impact of using agile methods on job satisfaction is the highest when one uses both agile PM and agile SDA practices. The direct impact of agile PM practices on job performance is notable but is complemented by the direct impact of both agile PM and agile SDA use constructs on job perception constructs. The agile PM constructs primarily impact the management side of the JCM (job autonomy and feedback), while the agile SDA constructs primarily impact the task side of the JCM (skill variety, task identity, and task significance). For managers, this finding suggests that implementing only one type of agile practice (SDA or PM) would have less impact on job satisfaction than implementing both types of practices. Therefore, management should be aware of the synergies realized from supporting and encouraging agile SDA and agile PM practices, specifically on perceptions of job autonomy, which is a strong predictor of job satisfaction.

Finally, our results should encourage agile practitioners to speak of agile development as more than a software-development method and instead as a way of organizing and designing work in a manner that can increase job satisfaction. Because satisfaction is a predictor of turnover, which reduces team performance (Argote, Insko, Yovetich, & Romero, 1995), management should attempt to increase job satisfaction. Agile practitioners can argue that using agile practices redesigns the work of software-development teams in a way that increases team members' positive perceptions of job characteristics. By underscoring the distinction among specific agile PM and agile SDA practices and their influence on job characteristics, our work affords opportunities for practitioners to justify including or excluding specific agile practices in how they organize new projects or address deficiencies in existing projects.

7 Limitations

This study has several limitations. First, with regard to the generalizability of our findings, we drew the sample from IS professionals working in the United States. Future research might consider cultural, geographical, and other differences to extend this study's generalizability across contexts. Additionally, we did not explore outsourcing, offshoring, and job status (i.e., contract labor vs. permanent employee) of ADT members. Second, the data were self-reported and the data-collection method was cross-sectional. Appendix B delineates two tests that suggest common method variance did not bias our results. Future research should gather longitudinal data to more rigorously test our theory.

Finally, although research has shown workplace characteristics to be an important factor affecting job satisfaction (McKnight, Phillips, & Hardgrave, 2009), we focused on the impact of agile development methods on job characteristics perceptions. For this reason, and because our survey instrument was already extremely long, we did not include workplace characteristics in our model. However, one could theorize both agile PM and agile SDA practices to impact workplace characteristics such as information sharing and trust in senior management (McKnight et al., 2009). This area is a potential opportunity for future research.

8 Conclusion

We developed a model of agile development practices' impact on ADT members' job satisfaction. We found mixed support for our hypotheses for the effect of agile practices use on job satisfaction and for the impact of agile practices use on job characteristics perceptions. This study demonstrates that IS-related organizational and process constructs relate to job characteristics perceptions. The agile SDA constructs are primarily related to the task side of job characteristics, while the agile PM constructs are primarily related to the management side of job characteristics. Astute managers should support and encourage both agile SDA and agile PM practices to maximize ADT members' job satisfaction.

Our model contributes to the agile software development and IT workforce literature by providing a theoretical lens that illustrates the pathways through which the use of agile methods impact job satisfaction, and it provides evidence of how Hackman and Oldham's (1980) job-design principles affect the perceptions of job characteristics.

References

Ahuja, M. K., Chudoba, K. M., Kacmar, C. J., McKnight, D. H., & George, J. F. (2007). IT road warriors: Balancing work-family conflict, job autonomy, and work overload to mitigate turnover intentions. *MIS Quarterly*, *31*(1), 1-17.

Anderson, D. J. (2004). *Agile management for software engineering: Applying the theory of constraints for business results*. New York: Prentice Hall.

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

Anderson, D. J. (2004). *Agile management for software engineering: Applying the theory of constraints for business results*. New York: Prentice Hall.

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

Angst, C. M., & Agarwal, R. (2009). Adoption of electronic health records in the presence of privacy concerns: the elaboration likelihood model and individual persuasion. *MIS Quarterly*, *33*(2), 339-370.

Argote, L., Insko, C. A., Yovetich, N., & Romero, A. A. (1995). Group learning curves: The effects of turnover and task complexity on group performance. *Journal of Applied Social Psychology*, *25*(6), 512-529.

Auer, K., & Miller, R. (2002). *Extreme programming applied*. New York: Addison-Wesley.

Ayyagari, R., Grover, V., & Purvis, R. (2011). Technostress: Technological antecedents and implications. *MIS Quarterly*, *35*(4), 831-858.

Balijepally, V., Mahapatra, R. K., Nerur, S. P., & Price, K. (2009). Are two heads better than one for software development? The productivity paradox of pair programming. *Management Information Systems Quarterly*, *33*(1), 91-118,

Beck. (2003). *Test-driven development: By example*. New York: Addison-Wesley.

Beck, K. (2000). *Extreme programming explained: Embrace change*. New York: Addison-Wesley.

Behson, S. J. (2012). Using relative weights to reanalyze research on the job characteristics model. *Journal of Organizational Psychology*, *12*(2), 71-81.

Benlian, A., Titah, R., & Hess, T. (2012). Differential effects of provider recommendations and consumer reviews in E-commerce transactions: an experimental study. *Journal of Management Information Systems*, *29*(1), 237-272.

Boehm, B., & Turner, R. (2004). *Balancing agility and discipline: A guide for the perplexed*. Boston, MA: Addison Wesley.

Boehm, B. W. (1984). Software engineering economics. *IEEE Transactions on Software Engineering, 10*(1), 4-21.

Burke, C., Stagl, K., Salas, E., Pierce, L., & Kendall, D. (2006). Understanding team adaptation: A conceptual analysis and model. *Journal of Applied Psychology*, *91*(6), 1189-1207.

Burke, R. J., & Greenglass, E. (1995). A longitudinal study of psychological burnout in teachers. *Human Relations*, *48*(2), 187-202.

Chin, W. W. (1998). Commentary: Issues and opinion on structural equation modeling. *MIS Quarterly*, *22*(1), vii-xvi.

Chin, W. W., Marcolin, B. L., & Newsted, P. R. (2003). A partial least squares latent variable modeling approach for measuring interaction effects: Results from a Monte Carlo simulation study and an electronic-mail emotion/adoption study. *Information Systems Research*, *14*(2), 189-217.

Chin, W. W., Thatcher, J. B., & Wright, R. T. (2012). Assessing common method bias: Problems with the ULMC technique. *MIS Quarterly*, *36*(3), 1003-1019.

Cockburn, A. (2001). *Agile software development*. Boston, MA: Addison-Wesley.

Cockburn, & Williams, L. (2001). The costs and benefits of pair programming. In G. Succi & M. Marchesi (Eds.), *Extreme programming examined* (pp. 223-248). New York: Addison-Wesley.

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences*. San Diego, CA: Psychology Press.

Conboy, K., & Fitzgerald, B. (2010). Method and developer characteristics for effective agile method tailoring: A study of XP expert opinion. *ACM Transactions on Software Engineering and Methodology*, *20*(1).

Cordery, J. L., Morrison, D., Wright, B. M., & Wall, T. D. (2010). The impact of autonomy and task uncertainty on team performance: A longitudinal field study. *Journal of Organizational Behavior*, *31*(2-3), 240-258.

Cox, K. B. (2003). The effects of intrapersonal, intragroup, and intergroup conflict on team performance effectiveness and work satisfaction. *Nursing Administration Quarterly*, *27*(2), 153-163.

Derby, E., Larsen, D., & Schwaber, K. (2006). *Agile retrospectives: Making good teams great*. Raleigh, NC: Pragmatic Bookshelf.

Deutsch, M. (1960). The effect of motivational orientation upon trust and suspicion. *Human Relations*, *13*, 123-139.

Dinger, M., Thatcher, J. B., Treadway, D., Stepina, L., & Breland, J. (2015). Does professionalism matter in the IT workforce? An empirical examination of IT professionals. *Journal of the Association for Information Systems*, *16*(4), 281-313.

Duvall, P. M., Matyas, S., & Glover, A. (2007). *Continuous integration: Improving software quality and reducing risk*. New York: Addison-Wesley.

Ferratt, T. W., Agarwal, R., Brown, C. V., & Moore, J. E. (2005). IT human resource management configurations and IT turnover: Theoretical synthesis and empirical analysis. *Information Systems Research*, *16*(3), 237-255.

Fornell, C., & Larcker, D. F. (1981). Evaluating structural equation models with unobservable variables and measurement error. *Journal of Marketing Research*, *18*(1), 39-50.

Fowler, M. (1999). *Refactoring. Improving the design of exiting code*. Reading, MA: Addison Wesley Longman.

Fowler, M. (2006). *Continuous integration*. Retrieved from http://www.martinfowler.com/articles/continuous integration.html

Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development*, *9*(8), 28-35.

Gefen, D., & Straub, D. (2005). A practical guide to factorial validity using PLS-Graph: Tutorial and annotated example. *Communications of the Association for Information Systems*, *16*, 99-109.

Grant, A. M., Fried, Y., & Juillerat, T. (2011). Work matters: Job design in classic and contemporary perspectives. In S. Zedeck (Ed.), *APA handbook of industrial and organizational psychology*. Washington, DC: American Psychological Association.

Griffeth, R. W., Hom, P. W., & Gaertner, S. (2000). A meta-analysis of antecedents and correlates of employee turnover: Update, moderator tests, and research implications for the next millennium. *Journal of Management*, *26*(3), 463-488.

Hackman, J., & Oldham, G. (1980). *Work redesign* (E. Schein & R. Bekhard, Eds.). New York: Addison Wesley.

Hackman, J. R., Brousseau, K. R., & Weiss, J. A. (1976). The interaction of task design and group performance strategies in determining group effectiveness. *Organizational Behavior and Human Performance*, *16*(2), 350-365.

Hair, J., Hult, G. T., Ringle, C., & Sarstedt, M. (2013). *A primer on partial least squares structural equation modeling (PLS-SEM)*. Thousand Oaks, CA: Sage.

Harris, M., Collins, R., & Hevner, A. (2009). Control of flexible software development under uncertainty. *Information Systems Research*, *20*(3), 400-419.

Harrison, D. A., Newman, D. A., & Roth, P. L. (2006). How important are job attitudes? Meta-analytic comparisons of integrative behavioral outcomes and time sequences. *Academy of Management Journal*, *49*(2), 305-325.

Highsmith, J. (2002). *Agile software development ecosystems*. Boston: Addison Wesley.

James, J. (2010). IT gender gap: Where are the female programmers? *TechRepublic.* Retrieved from http://www.techrepublic.com/blog/software-engineer/it-gender-gap-where-are-the-female-programmers/

Joseph, D., Ng, K.-Y., Koh, C., & Ang, S. (2007). Turnover of information technology professionals: A narrative review, meta-analytic structural equation modeling, and model development. *MIS Quarterly*, *31*(3), 547-577.

Karasek R. A., Jr. (1979). Job demands, job decision latitude, and mental strain: Implications for job redesign. *Administrative Science Quarterly*, *24*(2), 285-308.

Küster, J., Gschwind, T., & Zimmermann, O. (2009). Incremental development of model transformation chains using automated testing. In A. Schurr & B. Selic (Eds.), *MODELS 2009* (vol. 5795, pp. 733-747). Heidelberg: Springer.

Layman, L., Williams, L., & Cunningham, L. (2004). *Exploring extreme programming in context: An industrial case study* (pp. 32-41). Paper presented at the Agile Development Conference.

Lending, D., & Chervany, N. L. (2002). Case tool use and job design: A restrictiveness/flexibility explanation. *Journal of Computer Information Systems*, *43*(1), 81-90.

Lewicki, R. J., & Bunker, B. B. (1995). *Trust in relationships: A model of development and decline.* In B. B. Bunker & J. Z. Rubin (Eds.), *Conflict, cooperation, and justice: Essays inspired by the work of Morton Deutsch*. San Francisco: Jossey-Bass.

Lewis, J. D., & Weigert, A. (1985). Trust as a social reality. *Social Forces*, *63*(4), 967-985.

Liang, H., Saraf, N., Hu, Q., & Xue, Y. (2007). Assimilation of enterprise systems: The effect of institutional pressures and the mediating role of top management. *MIS Quarterly*, *31*(1), 59-87.

Mannaro, K., Melis, M., & Marchesi, M. (2004). Empirical analysis on the satisfaction of IT employees comparing XP practices with other software development methodologies. In *Extreme programming and agile processes in software engineering* (pp. 166-174). New York: Springer.

Mann, C., & Maurer, F. (2005). A case study on the impact of scrum on overtime and customer satisfaction. In *Proceedings of the Agile Conference* (pp. 70-79).

Maruping, L., Venkatesh, V., & Agarwal, R. (2009a). A control theory perspective on agile methodology use and changing user requirements. *Information Systems Research*, *20*(3), 377-399.

Maruping, L., Zhang, X., & Venkatesh, V. (2009b). Role of collective ownership and coding standards in coordinating expertise in software project teams. *European Journal of Information Systems*, *18*(4), 355–371.

Mathieson, K., Peacock, E., & Chin, W. W. (2001). Extending the technology acceptance model: The influence of perceived user resources. *ACM SigMIS Database*, *32*(3), 86-112.

Matook, S., & Maruping, L. M. (2014). A competency model for customer representatives in Agile software development projects. *MIS Quarterly Executive*, *13*(2), 77-95.

McHugh, C., K., & Lang, M. (2011). Using agile practices to influence motivation within it project teams. *Scandinavian Journal of Information Systems*, *23*(2), 85-110.

McHugh, O., Conboy, K., & Lang, M. (2010). *Motivating Agile teams: A case study of teams in Ireland and Sweden*. Presented at the 5th pre-ICIS International Research Workshop on Information Technology Project Management.

McHugh, O., Conboy, K., & Lang, M. (2012). Agile practices: The impact on trust in software project teams. *Software, IEEE*, *29*(3), 71-76.

McKnight, D. H., Phillips, B., & Hardgrave, B. C. (2009). Which reduces IT turnover intention the most: Workplace characteristics or job characteristics? *Information & Management*, *46*(3), 167-174.

Melnik, G., & Maurer, F. (2004). Direct verbal communication as a catalyst of agile knowledge sharing. In *Proceedings of the IEE Agile Development Conference* (pp. 21-31).

Melnik, G., & Maurer, F. (2006). Comparative analysis of job satisfaction in agile and non-agile software development teams. In *Extreme programming and agile processes in software engineering* (pp. 32–42). New York: Springer.

Moe, N., Dingsoyr, T., & Dyba, T. (2010). A teamwork model for understanding an agile team: A case study of a Scrum project. *Information and Software Technology*, *52*(5), 480-491.

Moore, J. E. (2000). One road to turnover: An examination of work exhaustion in technology professionals. *Management Information Systems Quarterly*, *24*(1), 141-168.

Morris, M. G., & Venkatesh, V. (2010). Job characteristics and job satisfaction: Understanding the role of enterprise resource planning system implementation. *MIS Quarterly*, *34*(1), 143-161.

Nahm, A. Y., Rao, S. S., Solis-Galvan, L. E., & Ragu-Nathan, T. S. (2002). The Q-sort method: Assessing reliability and construct validity of questionnaire items at a pre-testing stage. *Journal of Modern Applied Statistical Methods*, *1*(1), 114-125.

Naylor, J. C., Pritchard, R. D., & Ilgen, D. R. (1980). *A theory of behavior in organizations*. New York: Academic Press.

Nerur, S., Mahapatra, R. K., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM*, *48*(5), 72-78.

Nevo, S., & Chengalur-Smith, I. (2011). *Enhancing the performance of software development virtual teams through the use of aagile methods: A pilot study*. In Proceedings of the Hawaii International Conference on System Sciences (pp. 1-10).

Nunnally, J. C., Bernstein, I. H., & Berge, J. M. (1967). *Psychometric theory* (Vol. 226). New York: McGraw-Hill.

Palmer, S. P., & Felsing, J. M. (2002). *A practical guide to feature driven development*. Upper Saddle River, NJ: Prentice Hall.

Pavlou, P. A., Liang, H., & Xue, Y. (2007). Understanding and mitigating uncertainty in online exchange relationships: A principal-agent perspective. *Management Information Systems Quarterly*, *31*(1), 105–136.

Pedrycz, W., Russo, B., & Succi, G. (2011). A model of job satisfaction for collaborative development processes. *Journal of Systems and Software*, *84*(5), 739-752.

Petter, S., Straub, D., & Rai, A. (2007). Specifying formative constructs in information systems research. *MIS Quarterly*, *31*(4), 623-656.

Ply, J. K., Moore, J. E., Williams, C. K., & Thatcher, J. B. (2012). IS employee attitudes and perceptions at varying levels of software process maturity. *MIS Quarterly, 36*(2), 601-624.

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

Polites, G. L., Roberts, N., & Thatcher, J. (2012). Conceptualizing models using multidimensional constructs: A review and guidelines for their use. *European Journal of Information Systems*, *21*(1), 22-48.

Preacher, K. J., & Hayes, A. F. (2008). Asymptotic and resampling strategies for assessing and comparing indirect effects in multiple mediator models. *Behavior Research Methods*, *40*(3), 879-891.

Qureshi, I., & Compeau, D. (2009). Assessing between-group differences in information systems research: A comparison of covariance-and component-based SEM. *MIS Quarterly*, 197-214.

Ringle, C. M., Wende, S., & Will, A. (2005). *Smart PLS 2.0*. University of Hamburg. Retrieved from http://www. smartpls.de

Rutner, P. S., Hardgrave, B. C., & McKnight, D. H. (2008). Emotional dissonance and the information technology professional. *MIS Quarterly*, *32*(3), 635-652.

Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum*. Upper Saddle River, NJ: Prentice Hall.

Siponen, M., & Vance, A. (2010). Neutralization: New insights into the problem of employee information systems security policy violations. *MIS Quarterly*, *34*(3), 487-502.

Straub, D., Boudreau, M.-C., & Gefen, D. (2004). Validation guidelines for IS positivist research. *Communications of the Association for Information Systems*, *13*, 380-427.

Sutherland, J. (2001). Agile can scale: Inventing and reinventing scrum in five companies. *Cutter IT Journal*, *14*(12), 5-11.

Tessem, B., & Maurer, F. (2007). Job satisfaction and motivation in a large agile team. In G. Concas, E. Damiani, M. Scotto, & G. Succi (Eds.), *XP 2007* (vol. 4536, pp. 54-61). New York: Springer.

Thatcher, J. B., Liu, Y., Stepina, L. P., Goodman, J. M., & Treadway, D. C. (2006). IT worker turnover: An empirical examination of intrinsic motivation. *ACM SIGMIS Database*, *37*(2-3), 133-146.

Thatcher, J. B., Stepina, L. P., & Boyle, R. J. (2002). Turnover of information technology workers: Examining empirically the influence of attitudes, job characteristics, and external markets. *Journal of Management Information Systems*, *19*(3), 231-261.

Tripp, J. F., & Armstrong, D. J. (2014). Exploring the relationship between organizational adoption motives and the tailoring of agile methods. In *Proceedings of the 47th Hawaii International Conference on System Sciences* (pp. 4799-4806).

VersionOne. (2011). *6th annual state of agile development survey*. Retrieved from www.versionone.com

von Hippel, C., Kalokerinos, E. K., & Henry, J. D. (2013). Stereotype threat among older employees: Relationship with job attitudes and turnover intentions. *Psychology and Aging*, *28*(1), 17-27.

Vroom, V. H. (1964). *Work and motivation*. Oxford, England: Wiley.

Weiss, H. M., & Cropanzano, R. (1996). Affective events theory: A theoretical discussion of the structure, causes and consequences of affective experiences at work. In B. M. Staw & L. L. Cummings (Eds.), *Research in organizational behavior: An annual series of analytical essays and critical reviews* (vol. 18, pp. 1-75). New York: Elsevier.

West, D., Grant, T., Gerush, M., & D'Silva, D. (2010). Agile development: Mainstream adoption has changed agility. *Forrester Research*.

Williams, L. J., Edwards, J. R., & Vandenberg, R. J. (2003). Recent advances in causal modeling methods for organizational and management research. *Journal of Management*, *29*(6), 903-936.

Wright, R. T., Campbell, D. E., Thatcher, J. B., & Roberts, N. (2012). Operationalizing multidimensional constructs in structural equation modeling: Recommendations for IS research. *Communications of the Association for Information Systems*, *30*(1), 367-412.

Appendix A: Instrument

For the measurement items, we adapted some scales from previous research: we measured job satisfaction and job characteristics using scales slightly modified from Hackman and Oldham's (1980) scales (Morris & Venkatesh, 2010). To measure agile method use, we developed a new set of items. ADT members sometimes use similar terms to describe their practices even if those practices are not executed in the same manner. As such, we developed a scale that reflected practices in a manner consistent with the agile literature (e.g., Beck, 2000; Schwaber & Beedle, 2002). To develop this scale, one of the authors used agile development publications and the input of an external agile researcher and an agile practitioner to develop an initial set of items for each of the 10 agile practices of interest. After they developed these items, we followed the guidelines for applying a Q-sort (Nahm, Rao, Solis-Galvan, & Ragu-Nathan, 2002). We used two additional agile practitioners to perform a sorting exercise. Initial agreement on the scales was approximately 60 percent. We discussed the items that did not perform with the practitioners who did the sort. Based on this discussion, we modified or replaced the items, and three additional agile team members performed a second round of sorting. The Cohen's kappa for this sorting exercise was more than 85 percent, which indicates excellent agreement (Nahm et al., 2002). We used these items for the study.

Because research has shown job satisfaction to be one of the best predictors of turnover intention (Griffeth, Hom, & Gaertner, 2000), we examined several of the control variables that IT turnover studies have often used and chose age and organizational tenure (Ahuja et al., 2007; Joseph et al., 2007; Moore, 2000), negative affectivity (Moore, 2000) and gender and education (Joseph et al., 2007). Studies on job attitudes in the psychology literature have also used gender, tenure, and age as control variables (von Hippel, Kalokerinos, & Henry, 2013). Additionally, we also included total work experience as a control variable to account for the possibility that the culmination of a developer's total work experience might impact their job satisfaction.

For the sections of the survey that presented items relating to the JCM constructs and job satisfaction, the respondents were prompted with the statement: "For the following questions, please indicate to what extent YOU AGREE with the statements as they relate to your CURRENT job.". Because we sought to obtain feedback from development team members who used both agile and non-agile methods, we did not use the term "agile development" in the survey until after the respondents answered our questions regarding agile method use. Instead, we wrote the items in a manner that agile and non-agile team members could interpret, and we presented the respondent with the following prompt: "For the following questions, we would like you to think about THE WAY YOUR TEAM WORKS. When you see the term "work cycle", you should think about what your team uses as its shortest standard work cycle [we had defined this term previously] **If you see a specific term (such as "release"), please think about THAT term."

Table A1. Survey Instrument

Extent of agile practice use constructs Seven-point Likert scale ("strongly disagree" to "could agree or disagree" to and "strongly agree") and "don't know" as an eighth option.		
Construct	**Indicator**	**Item text**
1. Burndown	BD1	Our team utilizes visual indicators (charts, graphs, etc.) of how well we are progressing DURING a work cycle.
	BD2	We use visual tools that allow team members to easily tell if the work is being completed on schedule.
	BD3	We plot our work completed against work planned on a chart.
2. Iterative delivery	ID1	At the beginning of each work cycle, the team and business owners agree on what will be delivered during the work cycle.
	ID2	The team gives input as to how much work can be completed in a work cycle.
	ID3	The team estimates the amount of work each feature will require to be completed.
	ID4	Our team lets business people make business decisions about releases, and technical people make technical decisions about releases.
3. Stand-up	SU1	The team has a short meeting every day to discuss what is going on with the project.

Table A1. Survey Instrument

	SU2	Each day, all team members share with the team what they are working on.
	SU3	The team discusses issues together daily.
4. Retrospective	RE1	On a regular basis, the team reflects on previous work and looks for ways to improve team performance.
	RE2	At the end of each work cycle, the team asks itself "what went well" during the last work cycle.
	RE3	At the end of each work cycle, the team asks itself "what could be improved" during the next cycle.
5. Unit testing	UT1	We have a separate set of "test" code that is written specifically to test the "real" code.
	UT2	Every programmer is responsible for writing unit tests for the code he or she writes.
	UT3	Programmers are responsible to personally run a set of unit tests until they all run successfully before "checking in" changes.
6. Continuous integration	CI1	Members of the team integrate code changes as soon as possible.
	CI2	The team has a process that automatically rebuilds the software several times a day.
	CI3	The team is automatically notified of any issues related to the automated compiling, deployment or testing of code
7. Automated build	AB1	The team uses a script or other code to automatically compile the code.
	AB2	The team uses a script or other code to automatically build the software package.
	AB3	The team uses a script or other code to generate release notes or other documentation.
8. Coding standards	CS1	The naming and structure of our code is consistent.
	CS2	Our team uses standards for consistent code formatting.
	CS3	It is important to the team's success that all of the code be formatted consistently.
9. Refactoring	RF1	Whenever we see the need, we improve the design of the code we have written previously.
	RF2	Every member of the team attempts to improve the structure of the code when making a change.
	RF3	If we find code that is not used, we remove it.
10. Pair programming	PP1	When new software is being developed, two programmers concentrate on the code being written.
	PP2	We develop our code using pair programming.
	PP3	Our code is created by two people working together at a single computer.
Job characteristic model constructs Seven-point Likert scale (“strongly disagree” to “strongly agree”).		
Construct	**Indicator**	**Item text**
11. Feedback	FB1	Just doing the work required by the job provides many chances for me to figure out how well I am doing.
	FB2	After I finish a task, I know whether I performed well.
	FB3	The job itself provides you with information about your work performance.
12. Job autonomy	JA1	Your job gives you considerable opportunity for independence and freedom in how you do the work.
	JA2	Your job permits you to decide on your own how to go about doing the work.
	JA3	Your job gives you a chance to use your personal initiative and judgment in carrying out the work.

Table A1. Survey Instrument

	JA4	You have a great deal of autonomy in your job.
13. Skill variety	SV1	The job requires you to use a number of complex or high-level skills.
	SV2	The job requires you to do many different things at work, using a variety of your skills and talents.
	SV3	There is a great deal of variety in the work I perform.
14. Task identity	TI1	The job is arranged so that I can do an entire piece of work from beginning to end.
	TI2	The job provides me the chance to completely finish the pieces of work I begin.
	TI3	My job involves doing "whole" and identifiable pieces of work.
15. Task significance	TS1	The job itself is very significant and important in the broader scheme of things.
	TS2	The results of your work are likely to significantly affect the lives or well-being of other people.
	TS3	My job is very significant and important.
Job satisfaction Seven-point Likert scale ("strongly disagree" to "strongly agree").		
Construct	**Indicator**	**Item text**
16. Job satisfaction	JS1	Overall, I am satisfied with my job.
	JS2	I would prefer another, more ideal job.
	JS3	I am satisfied with the important aspects of my job.
Negative affectivity Five-point Likert scale ("not at all", "a little", "moderately", quite a bit", "extremely").		
Construct	**Indicator**	**Item text**
17. Negative affectivity	NA1	Right now, to what extent do you feel afraid?
	NA2	Right now, to what extent do you feel distressed?
	NA3	Right now, to what extent do you feel nervous?
	NA4	Right now, to what extent do you feel upset?
	NA5	Right now, to what extent do you feel ashamed?
	NA6	Right now, to what extent do you feel irritable?

Table A2. Respondent Breakdown by Team Role

Role	Number of respondents	Cumulative %
Team leader/project manager	88	31.0%
Architect	15	37.0%
Developer	133	89.9%
QA testing	18	96.8%
Business analyst	7	99.6%
Other	3	100.0%

Table A3. Summary of Agile Practice Use (VersionOne, 2011)

Rank	Practice	% of respondents
1	Daily stand-up	79%
2	Iteration planning*	74%
3	Unit testing	69%
4	Retrospectives	66%
5	Burndown	66%
6	Release planning*	65%
7	Velocity*	55%
8	Automated builds	55%
9	Continuous integration	55%
10	Coding standards	51%
11	Refactoring	48%
12	Test-driven development (TDD)	37%
13	Open work area	37%
14	Story mapping	34%
15	Digital taskboard	33%
16	Pair programming	29%
17	Collective code ownership	29%
18	Automated acceptance testing	25%
19	On-site customer	24%
20	Kanban	24%
21	Release-per-feature/continuous deployment	23%
22	Analog taskboard	23%
23	Agile games	16%
24	Cycle time (lean related)	11%
25	Behavior-driven development (BDD)	9%

Note: * practices merged into the iterative delivery concept
(Shaded rows are not included in our analysis)

Appendix B: Model Validation, Common Method Bias Testing, and Mediation Testing

To test convergent validity and reliability for the measurement model, we followed the recommended PLS validation procedures that Gefen and Straub (2005) outline. To test convergent validity, we performed a bootstrap with 200 resamples and then examined the outer model loadings. Convergent validity is demonstrated when all indicators load significantly on their respective latent construct. In our case, all indicators exhibited loadings that were significant at the .001 level (see Table B1), denoting strong convergent validity.

Table B1. T-statistics for Convergent Validity

Construct	Sub-construct (2nd-order weights, T-statistic)	Indicator	Outer loading	T-statistic
Agile PM practices	Burndown (.323, 17.882***)	BD1 ← burndown	.936	92.12***
		BD2 ← burndown	.943	112.70***
		BD3 ← burndown	.863	35.79***
	Iterative delivery (.330, 14.655***)	IP1 ← iterative delivery	.834	28.70***
		IP2 ← iterative delivery	.815	30.11***
		IP3 ← iterative delivery	.786	25.37***
		IP4 ← iterative delivery	.772	19.42***
	Retrospectives (.290, 14.584***)	RE1 ← retrospectives	.845	27.29***
		RE2 ← retrospectives	.919	64.47***
		RE3 ← retrospectives	.924	80.82***
	Daily stand-up (.306, 14.469***)	SU1 ← daily stand-up	.902	54.42***
		SU2 ← daily stand-up	.924	63.24***
		SU3 ← daily stand-up	.924	78.45***
Agile SDA practices	Automated build (.178, 9.315***)	AB1 ← automated build	.812	33.66***
		AB2 ← automated build	.868	33.35***
		AB3 ← automated build	.889	43.68***
	Continuous integration (.230, 12.038***)	CI1 ← cont. integration	.813	27.90***
		CI2 ← cont. integration	.882	42.44***
		CI3 ← cont. integration	.873	44.92***
	Coding standards (.253, 12.603***)	CS1 ← coding standards	.893	51.53***
		CS2 ← coding standards	.923	67.47***
		CS3 ← coding standards	.876	37.01***
	Pair programming (.210, 9.808***)	PP1 ← pair programming	.888	50.42***
		PP2 ← pair programming	.907	70.45***
		PP3 ← pair programming	.918	66.49***
	Refactoring (.245, 14.198***)	RF1 ← refactoring	.804	21.82***
		RF2 ← refactoring	.902	60.78***
		RF3 ← refactoring	.878	45.39***
	Unit testing (.206, 10.839***)	UT1 ← unit testing	.719	13.14***
		UT2 ← unit testing	.836	24.79***
		UT3 ← unit testing	.844	31.15***
Feedback	N/A	FB1 ← feedback	.862	38.20***
		FB2 ← feedback	.880	39.36***

Table B1. T-statistics for Convergent Validity

Construct	Sub-construct (2nd-order weights, T-statistic)	Indicator	Outer loading	T-statistic
		FB3 ← feedback	.900	55.03***
Job autonomy	N/A	JA1 ← job autonomy	.875	44.03***
		JA2 ← job autonomy	.847	34.66***
		JA3 ← job autonomy	.810	25.27***
		JA4 ← job autonomy	.802	22.29***
Skill variety	N/A	SV1 ← skill variety	.824	18.44***
		SV2 ← skill variety	.831	19.84***
		SV3 ← skill variety	.871	37.73***
Task identity	N/A	TI1 ← task identity	.905	58.04***
		TI2 ← task identity	.928	98.22***
		TI3 ← task identity	.891	48.15***
Task significance	N/A	TS1 ← task significance	.817	18.37***
		TS2 ← task significance	.833	31.90***
		TS3 ← task significance	.920	86.62***
Negative affectivity	N/A	NA1 ← negative affect	.769	16.22***
		NA2 ← negative affect	.785	21.16***
		NA3 ← negative affect	.828	28.61***
		NA4 ← negative affect	.789	18.34***
		NA5 ← negative affect	.714	15.12***
		NA6 ← negative affect	.725	14.12***
Job satisfaction	N/A	JS1 ← job satisfaction	.897	43.81***
		JS2 ← job satisfaction	.911	59.34***
		JS3 ← job satisfaction	.903	51.53***
Note: *** p < .001				

Another common test of convergent validity is to determine whether the average variance extracted (AVE) is at least .50 (Fornell & Larcker, 1981). In our model, the smallest AVE was .592. Both the Gefen and Straub (2005) and the Fornell and Larcker (1981) tests indicate a high degree of convergent validity.

In addition, to test the reliability of measurement items, we used SmartPLS to compute the Cronbach's α and a composite reliability score (Fornell & Larcker, 1981). All constructs exhibited a reliability score well above the threshold recommended for exploratory research (.60) (Nunnally, Bernstein, & Berge, 1967). Table B2 summarizes these results.

To evaluate discriminant validity, we performed two tests. First, we examined the cross-loadings of measurement items on their constructs. In this test, discriminant validity is demonstrated when an item loads more highly on its intended construct than on any other construct (Hair et al., 2013). An additional test suggests that the difference in loadings should exceed .10 (Gefen & Straub, 2005). In our case, all of our items passed both of these standard tests (see Table B3).

A third test of discriminant validity is to compare the AVE score for each construct. In the AVE test of discriminant validity, the square root of a given construct's AVE must be larger than the correlation of the construct with any construct in the model (Chin, 1998). All of our constructs passed this test, which indicates strong discriminant validity. Table B4 presents the correlations and square roots of the AVE.

Finally, to assess our aggregate second-order construct, we followed guidelines that Polites et al. (2012) and (Wright, Campbell, Thatcher, & Roberts, 2012) offer when evaluating the formative second-order agile PM practice and agile SDA practice factors. At the first-order level, Tables B1, B2, and B3 present evidence that the dimensions of agile PM and agile SDA factors exhibited both discriminant and convergent validity.

Consistent with an aggregate construct, per the explanations that Polites et al. (2012) offer, the respective first-order factors correlated at significant but varying levels. To assess the validity of the second-order factor, we assessed whether each dimension significantly correlated with the aggregate construct. We found that each first-order factor loaded significantly on the second-order factor. Table B1 lists the first-order factor loadings on the second-order factors.

Table B2. AVE and Reliability Scores

Construct	AVE	Composite reliability	Cronbach's α
Agile PM constructs			
Burndown	.837	.939	.902
Iterative delivery	.644	.878	.815
Retrospectives	.804	.925	.877
Daily stand-up meeting	.841	.941	.905
Agile SDA constructs			
Automated build	.735	.893	.819
Continuous integration	.734	.892	.818
Coding standards	.806	.926	.879
Pair programming	.818	.931	.889
Refactoring	.744	.897	.827
Unit testing	.643	.843	.719
Job characteristics model constructs			
	.776	.912	.856
Feedback	.696	.901	.815
Job autonomy	.709	.880	.805
Skill variety	.825	.934	.89
Task identity	.737	.893	.824
Task significance			
Negative affectivity	.592	.897	.861
Job satisfaction	.816	.930	.888

Table B3. PLS Factor Loadings and Cross Loadings

Construct	Item	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1. Burndown	BD1	**.94**	.48	.55	.51	.45	.52	.42	.43	.42	.52	.35	.15	.27	.31	.33	.38	-.21
	BD2	**.94**	.52	.51	.57	.53	.55	.46	.45	.44	.51	.40	.20	.30	.33	.32	.39	-.19
	BD3	**.86**	.49	.46	.49	.47	.55	.47	.45	.42	.52	.28	.05	.20	.31	.28	.23	-.10
2. Iterative delivery	IP1	.45	**.83**	.41	.43	.35	.35	.22	.38	.38	.28	.32	.23	.20	.32	.30	.36	-.15
	IP2	.48	**.82**	.33	.44	.30	.37	.26	.35	.42	.21	.26	.27	.29	.37	.25	.41	-.22
	IP3	.45	**.79**	.44	.40	.34	.37	.26	.31	.37	.35	.31	.19	.29	.18	.24	.28	-.06
	IP4	.36	**.77**	.38	.31	.34	.34	.22	.28	.36	.22	.25	.17	.19	.26	.20	.27	-.07
3. Stand-up	SU1	.52	.43	**.90**	.41	.35	.39	.31	.33	.36	.42	.31	.15	.12	.26	.18	.37	-.20
	SU2	.52	.44	**.92**	.44	.36	.44	.37	.31	.39	.48	.28	.11	.17	.24	.14	.29	-.14
	SU3	.49	.46	**.92**	.44	.41	.40	.33	.34	.41	.45	.35	.17	.23	.27	.18	.36	-.22
4. Retrospective	RE1	.52	.42	.43	**.85**	.52	.45	.38	.34	.44	.39	.36	.11	.23	.19	.26	.28	-.12
	RE2	.50	.44	.41	**.92**	.50	.46	.41	.40	.44	.35	.31	.13	.24	.21	.24	.34	-.10
	RE3	.52	.46	.43	**.92**	.45	.48	.44	.37	.38	.38	.31	.13	.24	.19	.21	.28	-.06
5. Unit testing	UT1	.43	.33	.36	.44	**.72**	.40	.45	.47	.43	.32	.26	.09	.08	.21	.20	.18	.01
	UT2	.42	.32	.33	.41	**.84**	.39	.38	.43	.44	.26	.36	.21	.20	.33	.29	.28	-.09
	UT3	.43	.35	.28	.45	**.85**	.42	.39	.45	.47	.30	.34	.15	.27	.28	.24	.25	-.08
6. Continuous integration	CI1	.48	.36	.41	.39	.41	**.81**	.52	.31	.39	.52	.31	.11	.25	.22	.24	.29	-.12
	CI2	.55	.40	.42	.47	.45	**.88**	.52	.42	.50	.43	.31	.14	.18	.24	.14	.31	-.15
	CI3	.50	.38	.32	.47	.43	**.87**	.60	.45	.50	.38	.24	.18	.17	.23	.17	.35	-.13
7. Automated build	AB1	.48	.21	.37	.45	.46	.57	**.81**	.36	.40	.51	.22	.02	.05	.17	.22	.17	-.01
	AB2	.37	.29	.26	.34	.46	.50	**.87**	.41	.33	.22	.18	.08	.15	.10	.16	.14	-.07
	AB3	.40	.29	.30	.37	.37	.56	**.89**	.33	.30	.23	.17	.11	.16	.16	.17	.18	-.04
8 Coding standards	CS1	.47	.40	.33	.38	.49	.44	**.34**	.89	.54	.34	.37	.20	.21	.35	.22	.31	-.06
	CS2	.47	.39	.36	.34	.51	.41	.36	**.92**	.54	.35	.33	.15	.18	.33	.30	.27	-.04
	CS3	.38	.33	.26	.40	.51	.40	.47	**.88**	.39	.31	.29	.17	.18	.31	.27	.26	-.01
9. Refactoring	RF1	.29	.33	.29	.31	.39	.38	.29	.39	**.80**	.38	.27	.22	.21	.27	.19	.23	-.05
	RF2	.45	.46	.36	.45	.52	.49	.36	.51	**.90**	.43	.32	.17	.24	.36	.27	.31	-.09
	RF3	.45	.43	.43	.43	.52	.52	.39	.51	**.88**	.39	.35	.18	.20	.36	.28	.33	-.08
10. Pair programming	PP1	.45	.28	.45	.34	.26	.44	.34	.27	.36	**.89**	.23	.06	.09	.20	.18	.24	-.08
	PP2	.53	.33	.40	.38	.36	.48	.34	.34	.46	**.91**	.29	.07	.12	.25	.21	.22	-.07
	PP3	.55	.28	.49	.40	.36	.47	.36	.39	.44	**.92**	.30	.00	.13	.25	.21	.20	-.14
11. Feedback	FB1	.31	.30	.32	.34	.30	.26	.18	.30	.33	.22	**.86**	.39	.32	.30	.28	.39	-.10
	FB2	.35	.30	.29	.29	.36	.31	.21	.34	.32	.24	**.88**	.39	.37	.34	.29	.36	-.16
	FB3	.33	.34	.31	.32	.39	.31	.20	.34	.32	.34	**.90**	.42	.35	.33	.38	.45	-.25
12. Job autonomy	JA1	.16	.26	.17	.13	.16	.16	.10	.17	.20	.06	.38	**.88**	.41	.53	.36	.55	-.18
	JA2	.14	.20	.08	.10	.14	.13	.06	.14	.18	.05	.33	**.84**	.40	.47	.32	.49	-.14
	JA3	.09	.24	.17	.14	.12	.15	.05	.17	.15	.00	.42	**.82**	.41	.41	.29	.47	-.11
	JA4	.10	.19	.09	.09	.20	.12	.04	.16	.20	.04	.39	**.80**	.48	.41	.34	.42	-.12
13. Skill variety	SV1	.19	.21	.12	.17	.15	.17	.08	.15	.13	.07	.28	.31	**.82**	.18	.43	.37	-.21
	SV2	.15	.21	.09	.13	.13	.10	.12	.13	.15	.05	.26	.38	**.83**	.16	.45	.32	-.12
	SV3	.32	.31	.23	.31	.27	.26	.14	.23	.31	.16	.40	.53	**.87**	.35	.51	.54	-.25
14. Task identity	TI1	.32	.28	.31	.21	.29	.26	.18	.33	.38	.29	.29	.49	.25	**.90**	.28	.44	-.20
	TI2	.32	.33	.25	.19	.35	.25	.16	.37	.37	.21	.34	.51	.25	**.93**	.28	.47	-.17
	TI3	.31	.36	.21	.19	.30	.22	.11	.31	.30	.21	.37	.50	.30	**.89**	.31	.49	-.19
15. Task significance	TS1	.19	.23	.08	.14	.20	.06	.10	.22	.19	.04	.25	.34	.46	.23	**.82**	.36	-.15
	TS2	.29	.23	.10	.21	.21	.22	.20	.21	.19	.18	.25	.30	.43	.17	**.83**	.33	-.19
	TS3	.36	.32	.24	.29	.34	.24	.23	.31	.33	.30	.39	.37	.53	.37	**.92**	.52	-.21
16. Job satisfaction	JS1	.39	.38	.36	.32	.28	.39	.22	.31	.33	.26	.44	.53	.40	.51	.45	**.90**	-.45
	JS2	.32	.39	.34	.30	.26	.33	.16	.27	.30	.19	.39	.53	.49	.45	.44	**.91**	-.31
	JS3	.28	.34	.30	.28	.27	.29	.13	.26	.28	.20	.39	.52	.50	.42	.43	**.90**	-.34
17. Negative affectivity	NA1	-.13	-.06	-.14	-.15	-.05	-.11	-.06	-.01	-.07	-.09	-.14	-.11	-.20	-.16	-.12	-.35	**.77**
	NA2	-.09	-.06	-.13	-.05	-.01	-.13	-.02	.03	-.08	-.07	-.18	-.19	-.17	-.14	-.18	-.30	**.78**
	NA3	-.20	-.15	-.17	-.14	-.06	-.12	-.11	-.06	-.07	-.12	-.14	-.14	-.25	-.19	-.21	-.32	**.83**
	NA4	-.11	-.17	-.17	-.04	-.06	-.09	-.01	-.02	-.05	-.07	-.13	-.11	-.17	-.11	-.16	-.26	**.79**
	NA5	-.09	-.05	-.06	.03	.01	-.03	.01	-.10	.01	.03	-.08	-.08	-.22	-.14	-.13	-.28	**.71**
	NA6	-.21	-.23	-.25	-.10	-.13	-.20	-.01	-.04	-.13	-.14	-.20	-.14	-.13	-.19	-.20	-.35	**.73**

Table B4. First-order Construct Correlations

Construct	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1. Auto. Build	**. 857**																
2. Burndown	.494	**.915**															
3. Continuous Int.	.635	.591	**.857**														
4. Coding Stand.	.430	.488	.464	**.898**													
5. Feedback	.227	.377	.335	.366	**.881**												
6. Iterative Plan.	.301	.545	.445	.416	.357	**.802**											
7. Job Sat.	.190	.366	.374	.312	.454	.413	**.904**										
8. Job Autonomy	.077	.149	.168	.190	.452	.268	.583	**.834**									
9. Negative Aff.	-.046	-.182	-.154	-.043	-.196	-.158	-.409	-.168	**.769**								
10. Pair Prog.	.383	.566	.516	.372	.307	.330	.246	.047	-.107	**.904**							
11. Refactoring	.405	.470	.542	.550	.367	.475	.341	.220	-.087	.465	**.862**						
12. Retrospective	.458	.575	.520	.415	.362	.493	.331	.140	-.104	.416	.467	**.897**					
13. Skill Variety	.140	.279	.230	.211	.391	.303	.513	.505	-.245	.126	.251	.265	**.842**				
14. Stand-up Mtg.	.367	.556	.448	.354	.345	.483	.369	.154	-.203	.493	.423	.472	.188	**.917**			
15 Task Sig.	.215	.338	.212	.295	.359	.311	.486	.395	-.217	.222	.290	.263	.555	.179	**.858**		
16 Task Identity	.168	.348	.267	.368	.368	.356	.514	.550	-.205	.260	.386	.219	.296	.282	.317	**.908**	
17. Unit Testing	.508	.530	.504	.561	.399	.415	.297	.186	-.066	.364	.557	.544	.235	.404	.305	.345	**.802**
Note: Diagonal contains square root of AVE																	

Tests for Common Method Bias

To test for common method bias, we first performed Harman's single-factor test (Podsakoff, MacKenzie, Lee, & Podsakoff, 2003). In this test, one enters all items into an unrotated exploratory factor analysis to determine whether a single factor emerges or a single factor accounts for the majority of the variance. In our test, 37 factors emerged, the largest of which accounted for 36.8 percent of the variance. This result was the first evidence that common method bias was not an issue.

We also performed the correlation test from (Pavlou, Liang, & Xue, 2007). We found that no constructs correlate extremely highly (more than .90). This finding was the second evidence that common method bias was not a problem in our study.

We performed the test that Liang, Saraf, Hu, and Xue (2007) suggest even though some have raised issues with this test (Chin, Thatcher, & Wright, 2012). In this test, one creates a single item factor for each latent indicator, and one creates a single "method" factor that contains all of the indicators. One then runs the model with these factors[6]. Table B5 presents the results of this test. One identifies common method bias by examining the statistical significance of the loadings of the single-item constructs and both the "substantive" constructs and the "method" construct. Further, one compares the variance explained by the substantive factors versus the variance explained by the method factor (Williams, Edwards, & Vandenberg, 2003). The percentage of variance explained is the square of the loadings. If the method construct loadings are generally insignificant, and the percentages of indicator variance explained by substantive constructs are substantially greater than those explained by the method construct, then common method bias is demonstrated to have minimal effect and, thus, be of little concern.

As Table B5 shows, variance of indicators caused by substantive constructs was substantially greater than that caused by the method construct. The average variance caused by substantive constructs was 74 percent compared with less than 1 percent for the method construct. Further, when considering the loadings of the method factor, we found that the large majority of factors were insignificant. We conclude that there was little influence of common method bias.

[6] For an excellent summary of this technique and how to complete it, see Siponen and Vance (2010).

Table B5. Common Method Bias Analysis

Construct	Indicator	Substantive factor loading	Variance explained	Method factor loading	Variance explained
1. Burndown	BD1	**.95****	.90	-.02	.000
	BD2	**.91****	.83	.05	.003
	BD3	**.87****	.76	-.03	.001
2. Iterative delivery	IP1	**.82****	.67	.02	.000
	IP2	**.78****	.61	.05	.003
	IP3	**.77****	.59	.02	.000
	IP4	**.84****	.71	-.09	.008
3. Stand-up	SU1	**.91****	.83	-.01	.000
	SU2	**.94****	.88	-.02	.000
	SU3	**.91****	.83	.03	.001
4. Retrospective	RE1	**.80****	.64	.07	.005
	RE2	**.93****	.86	-.02	.000
	RE3	**.96****	.92	-.04	.002
5. Unit testing	UT1	**.63****	.40	.08	.006
	UT2	**.88****	.77	-.03	.001
	UT3	**.88****	.77	-.03	.001
6. Continuous integration	CI1	**.80****	.64	.02	.000
	CI2	**.88****	.77	.005	.000
	CI3	**.89****	.79	-.02	.000
7. Automated build	AB1	**.71****	.50	.13*	.017
	AB2	**.92****	.85	-.06	.004
	AB3	**.94****	.88	-.05	.003
8. Coding standards	CS1	**.89****	.79	.02	.000
	CS2	**.92****	.85	.005	.000
	CS3	**.88****	.77	-.02	.000
9. Refactoring	RF1	**.90****	.81	-.13*	.017
	RF2	**.88****	.77	.03	.001
	RF3	**.81****	.66	.08	.006
10. Pair programming	PP1	**.93****	.86	-.05	.003
	PP2	**.89****	.79	.02	.000
	PP3	**.90****	.81	.03	.001
11. Feedback	FB1	**.88****	.77	-.03	.001
	FB2	**.89****	.79	-.01	.000
	FB3	**.87****	.76	.04	.002
12. Job autonomy	JA1	**.85****	.72	.04	.002
	JA2	**.85****	.72	-.03	.001
	JA3	**.82****	.67	-.005	.000
	JA4	**.82****	.67	-.01	.000
13. Skill variety	SV1	**.90****	.81	-.07*	.005
	SV2	**.93****	.86	-.121*	.015
	SV3	**.71****	.50	.20*	.040

Table B5. Common Method Bias Analysis

Construct	Indicator	Substantive factor loading	Variance explained	Method factor loading	Variance explained
14. Task identity	TI1	**.90****	.81	.008	.000
	TI2	**.93****	.86	-.009	.000
	TI3	**.89****	.79	.0009	.000
15. Task significance	TS1	**.90****	.81	-.11*	.012
	TS2	**.88****	.77	-.05	.003
	TS3	**.81****	.66	.15*	.023
16. Job satisfaction	JS1	**.82****	.67	.10	.010
	JS2	**.93****	.86	-.03	.001
	JS3	**.96****	.92	-.07	.005
17. Negative affectivity	NA1	**.76****	.58	-.003	.000
	NA2	**.80****	.64	.03	.001
	NA3	**.82****	.67	-.04	.002
	NA4	**.82****	.67	.03	.001
	NA5	**.74****	.55	.08	.006
	NA6	**.67****	.45	-.10	.010
Average		**.86**	.74	.00	.00
Note: ** - p < .01; *- p <.05					

About the Authors

John F. Tripp is an Assistant Professor of Information Systems at Baylor University. Before beginning his PhD, he worked for more than 17 years in industry as a software developer, project manager, and IT Director. His research has appeared in the *Journal of the Association for Information Systems*, *Information Systems and e-Business Management,* and the *Journal of Management Systems*. He currently examines topics such as agile software development, social networking, trust and privacy, and how technology influences leadership, collaboration, and communication in organizations.

Cynthia K. Riemenschneider is associate dean for research and faculty development and professor of Information Systems in the Hankamer School of Business at Baylor University. She also holds the Helen Ligon Professorship in Information Systems. Her publications have appeared in *MIS Quarterly, Information Systems Research, Journal of Management Information Systems, Information Systems Journal, European Journal of Information Systems, IEEE Transactions on Software Engineering*, and others. She currently conducts research on IT work force issues, women and minorities in IT, and ethical issues surrounding IT use.

Jason Thatcher is a Professor of Information Systems at Clemson University. His research examines drivers of technology use in organizations, including individual innovation with technology, management of the IT workforce, and organizations' IT strategy. He work appears in *MIS Quarterly, Information Systems Research, Journal of Applied Psychology*, and other outlets. He is actively involved in service to the IS discipline. He is President-Elect of the Association for Information Systems. He serves as a Senior Editor at *MIS Quarterly*, *Decision Sciences*, *AIS Transaction on Human Computer Interaction*, and other refereed outlets. He has also served as an editorial board member of *Information Systems Research*, *Journal of the Association for Information Systems*, and *IEEE Transactions on Engineering Management*.