ABSTRACT

This study assesses software professionals’ job perceptions in organizations adopting different software process models. We developed hypotheses grounded in the formal control and the informal control theories. We plan to conduct a survey of software professionals in organizations utilizing different process models. Theoretical and practical implications of this research are discussed.

Keywords

Software process models, agile, plan-driven, job perceptions

INTRODUCTION

Developing software to meet the highly dynamic requirements of the twenty-first century remains a difficult task. Rapid changes in technology, business models, and customer needs continue to press for quick releases of new software products. It's a given that these products meet the necessary quality constraints. Software products range from very small stand-alone systems, e.g. a simple cell-phone application, to highly integrated software such as ERP systems. Customer sets vary accordingly. Software development teams are not all the same either. The demands of the product developed require different team characteristics with a range of technical ability, software method understanding, and individual empowerment. Understanding the entirety of these constraints, software development organizations are confronted with the decision on how best to manage the software development process. There are currently two schools of thought: use traditional plan-driven processes or use agility-based agile processes.

Making the software process choice may be guided by five critical project factors (Boehm, 2004): size, criticality, dynamism, personnel, and culture. All five factors challenge our decision. Of these, perhaps the most difficult ones to consider are the factors that are addressed by an organization's personnel and culture. Selecting an agile process requires an organization that empowers people to move in and out of different roles in order to solve problems quickly and efficiently whereas if a plan-driven process is selected, roles are typically well defined with a high degree of structure.

Having a high degree of freedom might not work well for some personnel. Role ambiguity and conflict might arise more easily in one software development process than another. Some developers may take on work that is beyond what they can accomplish for what they perceive to be moving the project forward. This could lead to work overload and exhaustion. Cynicism may creep in when developers are relegated to strict roles. Too much or too little individual or team autonomy may also affect the project. Finally, the process choice may influence one's perceived effectiveness and accomplishments as well as overall job satisfaction. It is these questions that our research intends to address.

The following research question is posed: “Do software professionals’ job perceptions differ in organizations adopting different software process models? In particular, is using an agile software development process model related to more favorable job perceptions?”
In order to investigate this question, we leverage a theoretical framework based on control theory. The results of the study are critical from both theoretical and practical standpoints. From a theoretical viewpoint, the proposed study extends the current knowledge base with respect to job perceptions in plan-driven vs. agile software process environments. Practically, a better understanding of the various dimensions and how they differ will allow organizations to design mechanisms to improve employees’ job perceptions.

The paper is organized as follows. The next section describes different software development process models. We then present the theoretical framework and propose the hypotheses. We discuss the data collection and analysis method. The theoretical and practical implications of this study are presented as well.

SOFTWARE DEVELOPMENT PROCESS MODELS: PLAN-DRIVEN VERSUS AGILE

Frederick Brooks (Brooks, 1987) published a seminal paper regarding the challenges of software engineering. He identified four inherent properties of modern software systems, i.e. complexity, conformity, changeability, and invisibility that drive these challenges. Requirements refinement and incremental development was suggested, among others, as one of the potential breakthroughs in software engineering.

Parnas and Clements (1986) provided a description of a rational design process, also referred to as the waterfall model, for software development. The rational design process included seven sub-processes, or stages: (a) requirements definition, (b) module design, (c) interface design, (d) hierarchy design, (e) internal design, (f) program implementation, and (g) maintenance. For each phase a set of deliverables are defined and the people who would perform the tasks are identified. Before moving on to the next phase, the deliverables must be accepted by the customer. Parnas and Clements' rational design process characterizes a plan-driven approach to software development. Since the publication of their paper, numerous modifications and adaptations of their design process have come of age. However those that adhere closely to this type of design process are typically characterized as process focused, driven by documentation and customer contracts, and plan-driven, all of which are the reasons why those who back the agile manifesto (2001) might choose not to use a rational-design process.

The agile manifesto (2001) provides an official definition of Agile Software Development reading as follows: Individuals and interactions over processes and tools, Working software over comprehensive documentation, Customer collaboration over contract negotiation, and Responding to change over following a plan. Twelve principles are listed that reflect a development methodology that must be highly adaptable and nimble for it to succeed. Dingsøyr et al. (2012) provide a summary overview of the historical research on agile software development. Although organizations might wish to move towards an agile methodology, changing methodologies, especially ones that require a drastic change in philosophy, is no simple matter. Chan and Young (2009) describe the barriers to acceptance of software development methodologies. They provide a conceptual framework that might be used to better understand the acceptance of agile methods. Nerur, RadhaKanta, and Mangalaraj (2005) describe key issues in the migration to an agile methodology categorizing them as management and organizational, people, process, and technology. An argument for the method of choice might best be determined by the risk associated with the use of the particular methodology.

Boehm (2002) advocates for a synthesis of agile and plan driven development. The idea of bringing best practices to the table is not a new idea. Humphrey's (1988) software process maturity framework incorporates the use of best practices. Boehm lists several project characteristics including developers that influence the choice of a development methodology. Generally the research suggests that there is no one methodology that works for all software development projects. Organizations that incorporate the best of plan-driven and agile methods based on risk and opportunity will typically win the software development challenge.

THEORETICAL FOUNDATION: CONTROL THEORY

Control is defined as management’s “attempts to ensure that individuals working on organizational projects act according to an agreed-upon strategy to achieve desired objectives” (Kirsch, 1996 p.1). Control theory was first conceptualized by Ouchi (1979). Ouchi describes three control mechanisms: markets, bureaucracies, and clans. In market control, controllers rely on market prices to exercise control on the controlees. Ouchi used an example of a purchasing department to illustrate the point – purchasing agents send out a description of the item desired and ask each manufacturer to quote a price for it; after the prices are in, the purchaser makes a decision based on quotes and past performance of the manufacturers. Ouchi argues since
prices convey all of the information necessary for efficient decision-making, market control is a very efficient mechanism. Bureaucratic mechanisms, on the other hand, involve personal surveillance and supervisory directions to the subordinates. Therefore, it requires rules and legitimate authority to be functional. Ouchi states that even though it is not as efficient as the market control, the bureaucratic form is preferred when prices are not available or infeasible to measure. Ouchi takes a warehouse example to support his viewpoints. In a warehouse, the foreman sees workers follow rules to fill, pack, and ship the order. Ouchi describes a clan as different organizations of professions but with similar values, brought together by socialization processes. Clan control relies on informal social structure, shared values, and beliefs. Ouchi cites the general hospital as a clan example.

Eisenhardt (1985) extended Ouchi’s control theory by incorporating agency theory. She described organizational control theory and agency approaches as complementary where organizational control emphasizes task characteristics, e.g. task programmability, to the choice of control strategy, and the existence of social control as an alternative while agency theory adds to the organizational approach explicit emphases on information systems, business uncertainty, costs, and rewards (p.7). She hypothesized linkages of task characteristics, information systems (measurability of behavior and outcome) and business uncertainty to behavior and outcome control strategies. She studied 54 retail stores. She found behavior based control is more likely when task programmability increases, behavior measurement increases, cost of outcome measurement increases, and business uncertainty increases. Among all the variables, task programming is the single most important predictor.

Control theory has been applied to investigate software development phenomena (e.g. Henderson and Lee, 1992; Kirsch, 1996, 1997; Choudhury and Sabherwal, 2003; Harris, Collins, and Hevner, 2009; Maruping, Venkatesh, and Agarwal, 2009; Ply, Moore, Williams, and Thatcher, 2012). These studies generally categorize controls into formal and informal controls, with formal controls initiated by management, and informal controls by employees (Jaworski et al., 1993). Formal controls are defined as using formal documentation associated with performance evaluation and rewards to influence the personnel’s behavior to support organizational objectives (Choudhury and Sabherwal, 2003; Kirsch, 1996; Ply et al., 2012). Formal controls include behavioral and outcome. Behavioral controls rely on rules and procedures that lead to desired outcomes, and the ability to observe the associated behaviors (Ply et al., 2012). Outcome controls focus on the specification of desired outputs and rewards for their achievement (Kirsch 1996, 1997). The two types of informal controls are clan control and self control. Clan control leverages socialization processes, shared values, and beliefs to influence the behavior of individuals, (Kirsch, 1996, 1997). With self control, individuals control their own actions: they set goals, monitor work, and reward or sanction another (Kirsch, 1997).

Among the control theory based information systems studies, two examined areas are directly relevant to our research: one investigated the interactions between control modes, agile methodology use, and requirements change (Maruping et al., 2009), and the other examined IS employee attitudes and perceptions related to software process maturity levels (Ply et al., 2012). Maruping et al. (2009) concluded control mechanisms moderate the relationship between agile methodology use and project quality. In particular, agile methodology use leads to higher quality when outcome control is high and self control is low. Ply et al. (2012) found employees at CMM level 3 where behavioral controls are dominant have lower professional efficacy and lower job satisfaction than in organizations at Level 1 which is relatively free of formal controls, and when behavioral controls are supplemented by substantial outcome controls, in cases of CMM Levels 4/5, employees reported higher job satisfaction than their counterparts in organizations at CMM Level 3.

Relating control theory to agile and plan-driven software process environments, we argue both software processes apply outcome control due to the nature of software business. Any software, when delivered, needs to meet minimum user expectations, meaning it needs to provide the required features and to perform reasonably. This outcome can be observed when developers test the software and the users use the software. We also argue in agile the outcome measures utilized for evaluation are less blurry than in plan-driven. In agile software development, the focus of the outcome is “working software” and the outcome measure is negotiated and made known to all parties ahead of time. In plan-driven software development, measures such as process improvement, design qualities, and documentation are applied along with working software. Some measures are for developers only and others are for both developers and users. They are not necessarily consistent all the time.

Besides outcome control, based on the unique characteristics of the two software processes, we argue agile leans towards clan control while plan-driven adopts behavioral control. In agile the manager is a facilitator, in plan-driven the manager is a controller; agile encourages collaboration and communication, plan-driven manages through control and direction; agile takes a flexible approach adapted with collective understanding, plan-driven adopts a universal approach; agile relies on self-organizing teams and face-to-face communication, plan-driven controls at the organizational level and depends on document-based communication (Dyba and Dingsøyr, 2009; Salo and Abrahamsson, 2007). Clan control in agile is demonstrated.
through flexible management style, collective decision making, team-oriented communication and working environment. Behavioral control in plan-driven is illustrated by top-down management style, individualized task assignments, and focuses on processes and documentations.

HYPOTHESES

In this section, we develop hypotheses on constructs that are related to job perceptions salient to software development professionals. The constructs are role ambiguity, role conflict, work overload, work exhaustion, cynicism, autonomy, professional efficacy, and job satisfaction. Previous studies suggest these constructs are important for studying job attitudes and job perceptions of software development professionals (e.g. Ply et al., 2012).

Role ambiguity

Role ambiguity occurs when an individual does not have access to role-related information that defines their job behavior expectations (Kahn, Wolfe, Quinn, Snoek, and Rosenthal, 1964, p. 22). Role-related information may be communicated in various ways, formally, or informally. Bauer and Simon (2000) provide a review of the generally accepted definitions of role ambiguity and provide four widely accepted dimensions including expectation, process, priority, and behavior. Boehm (2004) contrasts the cultural differences between agile and plan-driven development. In particular, a team member in an agile culture acts as a craftsman with no formal roles, taking on any task that moves a project forward. In a plan-driven culture, roles and tasks are well defined. However, one must also consider that the personnel typically involved across the two software process models are different. Agile process models work best when there is a critical mass of highly talented people (Boehm, 2004, p. 54; Nerur, Mahpatra, and Mangalaraj, 2005).

In a plan-driven model, a work breakdown structure is developed; each task has a specific duration, task dependencies, and resource requirements. Resource requirements are typically human; where role specifics are outlined and individuals are identified to fit into these roles. These constraints are placed into a project planning application that may be used to optimize the critical path for the completion of the project. Agile methods would spend less time in the process of developing an entire schedule, with focus on the next product component to be released. Continuous planning occurs in various degrees, built into release schedules that may be daily, weekly, or monthly. Human resources are more dynamic; teams are self-organizing filling in the roles of systems analyst, developer, tester, and documentation specialist with members of the team as they fit best. From one release cycle to the next, team members may take on completely different roles. It is this fluidity of role responsibility that is much different than what occurs in a plan-driven model, and may cause uncertainty in work behavior for an individual.

We hypothesize:

H1: IS professionals in agile software process environments will report higher role ambiguity than those in plan-driven software process environments.

Role conflict

Role conflict is the perception of inconsistent or incompatible job expectations (Nelson and Quick, 2003). We suspect role conflict will be higher for IS professionals in agile than for IS professionals in plan-driven.

In plan-driven software development, the developer generally plays a distinct role. The developer knows exactly what he or she needs to do, primarily interacting with his/her supervisor. The supervisor assigns tasks, knows deadlines for the tasks, and does performance evaluation on the developer. Keeping the supervisor in the loop is sufficient for the developer.

In agile software development, developers work in a team environment. The work assignment can come from their supervisors, from the users directly, or could come from co-workers who may need the developer's help on some programming task. The supervisor does the performance evaluation on the developer, but maybe the person who knows how the developer performed best is one of the end users or the project manager. The developer is pulled from different directions: the supervisor, the project manager, the user, and co-workers. Each of these groups may have inconsistent or incompatible expectations of the developer. Keeping the supervisor in the loop is far from sufficient.

Because of the above reasons, we hypothesize:
H2: IS professionals in agile software process environments will report higher role conflict than those in plan-driven software process environments.

Work overload

Work overload is a situation in which someone has too much work to do (Cambridge, 2012). Work overload can be caused by a variety of factors. Examples are unrealistic deadlines and increasingly heightened expectations (Shimazu and Kosugi, 2003). Gryna (2004) describes common symptoms of work overload being long work days, unwanted overtime, difficulty in taking vacation time, taking work home, and frequent “firefighting”.

In plan-driven software development, crunch time (periods of extreme work overload occurring usually closer to deadlines) is a common and recurring theme. The image of professional developers is a one with sleeping bags in the office who burns midnight oil. Meeting user expectations and delivering projects on time has become challenging as users demand more reduced delivery times and more trustworthy software (Sommerville, 2011).

Adopting the agile principles is supposed to address the above problems. Cooke (2010) describes three (along with others) agile principles. First is the communication between developers and end users. Constant communication helps all parties involved understand what needs to be done and when it will be done. It minimizes unexpected demands on the tasks. Second is the empowerment of agile teams to determine the volume of work that they are capable of delivering. The teams are given a sense of responsibility for completing the assignment within the timeframe they have set themselves. Third is the use of iterative development which avoids the risks involved in all-at-once delivery. Agile promotes sustainable development and the practice of the 40 hour work week.

We hypothesize:

**H3: IS professionals in agile software process environments will report lower work overload than those in plan-driven software process environments.**

Work exhaustion

Pines et al. (1981) defines work exhaustion as the result of constant or repeated emotional pressure associated with an intense involvement with people over long periods of time. Maslach and Jackson (1981, 1986) operationalize work exhaustion as a three-component model: psychological syndrome of emotional exhaustion, depersonalization, and diminished personal accomplishment. In our paper, we focus on the first component – emotional exhaustion.

Several factors contribute to emotional exhaustion: intense concentration on a task without appropriate breaks, disordered multitasking, interruptions during work, and chaos in the workplace (Zohar, 1999).

One of the principles behind the Agile Manifesto is “Business people and developers must work together daily throughout the project” (Agile Principles, 2013). Communications between developers and business people bring many benefits undoubtedly, but at the same time could be challenging due to several reasons. First, developers and business people are trained in different fields, each with its own distinct sets of vocabularies that are hard to convey to outsiders. Secondly, technically trained professionals are typically good at logic and hands-on tasks, some of them may not be comfortable with constant face-to-face verbal communications. Thirdly, the devotion and the patience of the business people on the technical component of the software project might be limited since they have their own jobs to be concerned about. All these could lead to emotional exhaustion.

The work space for agile teams is typically an open area with desks, chairs, and computers. Users are encouraged to come by to communicate with the developers. The developers are encouraged to interrupt each other with questions and discussions. Compared to plan-driven environments where developers work in their own cubicles and offices, an environment usually exclusive from the users, the agile work environment has more noises and interruptions. Furthermore, the daily planning in agile suggests developers may need to switch between tasks frequently.

Pair programming is one of the agile practices. In pair programming, two programmers sit in front of one computer and work on the same programming task together. One person is the driver, and the other is the navigator. The driver controls the keyboard, and the navigator examines what is being typed in and think of other strategies to accomplish the task. The two change roles periodically. Previous research show pair programming could lead to fatigue. A pairing session of no more than two to three hours is recommended, because after that, fatigue can set in (agilesherpa, 2013).
We hypothesize:

\[ \text{H4: IS professionals in agile software process environments will report higher work exhaustion than those in plan-driven software process environments.} \]

**Cynicism**

Cynicism reflects doubt, disbelief, and distrust. Kanter and Mirvis (1989) draw on their national survey of American employees and reveal the causes of cynicism at work. The causes are: a lack of meaningful and challenging work, limited opportunity for advancement, and negative managerial style. They stated building a sense of community was crucial and could be accomplished through communication strategies, evaluation procedures, and other innovations. Our discussions below focus on meaningful work, managerial style, and a sense of community assuming similar challenging levels of work and opportunity for advancement are present in both agile and plan-driven environments.

In plan-driven software development, communications generally happen through written documentations instead of verbal conversations. Except for the few lines on the request form, developers don’t have to hear from the users regarding why they need the system and how the system is going to help with their work. The meaning of work is greatly reduced because of this communication style. In addition, since most requests are handled through the request tracking system, there are fewer chances for users and developers to interact, therefore hard to develop a sense of community. Regarding managerial style, plan-driven organizations typically adopt behavior control. Managers focus on the processes the developers follow, and rely on intermediate reports to assess its progress. A certain degree of outcome control might be used, but in plan-driven, since users and developers have limited communications, the outcome can be harder to measure. Different angles the users and the developers take can easily lead to different measures of the outcome, and it is difficult to determine which one should be used and which one is more important.

Agile software development encourages collaborations. Users and developers meet on a regular basis. In some agile practices, e.g. Scrum, the team holds a stand-up meeting every day. Everyone knows what he/she needs to work on, the reason for the task, and when it needs to be done. Team members are encouraged to ask questions and to seek for clarifications. If there are issues and questions during the day, users can come in freely, and developers are free to interrupt each other, so questions are answered, issues are addressed, and there is increased personal contact between everyone involved in the project. The information the users share with the developers brings meaning to their request, and the constant communications between users and developers and among developers bring a strong sense of community. In addition, agile adopts incremental development and constant feedback, which suggests even when outcome control is applied, the developers and users tend to agree on the measures. Agile also adopts clan control - team-monitoring, which leads to a better sense of community.

Therefore, we hypothesize:

\[ \text{H5: IS professionals in agile software process environments will report lower cynicism than those in plan-driven software process environments.} \]

**Autonomy**

Autonomy means independence and self-governing. In software development, autonomy refers to the degree of discretion and independence granted to individuals or teams in scheduling the work, determining the procedure and methods to be used, selecting and deploying resources, and carrying out tasks (e.g. Lee and Xia, 2010).

In plan-driven software development, a request is entered into a system, the request is processed, and based on the priorities, is assigned to a developer. Even though the user specifies on the request form when he/she needs the request to be completed, in general it is the developer who works on the task that determines the methods he/she needs to adopt, the resources and the time he/she will need to accomplish the task. On the other hand, agile development encourages collaboration and is people-centric. One individual usually doesn’t make a decision. Decisions are made as a team. For example, when the team has the daily stand-up meetings, decisions are negotiated and finalized during the meetings regarding who will work on what, what resources are needed for a given task, and when task will be completed.

While autonomy is a key principle in agile development, empirical research has been limited. Lee and Xia (2010) examined team autonomy and found a negative relationship between autonomy and team response extensiveness and a positive relationship between autonomy and team response efficiency. Moe et al. (2008) researched on barriers of introducing self-
organizing teams in agile and found that highly specialized skills of the developers, the corresponding division of work, lack of system for team support, and reduced external autonomy were important barriers.

We hypothesize:

**H6:** IS professionals in agile software process environments will report higher team autonomy but lower individual autonomy than ones in plan-driven software process environments.

**Professional efficacy**

Professional efficacy is an individual’s perceived effectiveness and accomplishment at work (Schaufeli et al., 1995).

Research on IS employees’ professional efficacy is rather limited. The only one we identified is Ply et al. (2012). The authors examined professional efficacy in organizations at different CMM levels. They found IS workers reported significantly lower professional efficacy in organizations at CMM Level 3 than in organizations at Level 1 because of formal controls in CMM Level 3 organizations.

In the context of agile and plan-driven development, one could argue employees in agile organizations will report higher professional efficacy than employees in plan-driven development based on the underlying differences between agile and plan-driven. In plan-driven development, behavioral control is the primary control mechanism. The supervisor assigns the work and provides direct supervision to the employees. Formal documentations are the primary communication tools. Feedback from the users is rather limited. A formal request tracking system is usually in place and serves as the communication channel for developers and users. In agile development, the control style is primarily clan control – the team makes the decision. Constant and immediate feedback from users improves the sense of effectiveness and accomplishment. With iterative development, even if there are issues, they can be identified quickly, and corrections can be applied without hurting the sense of effectiveness and accomplishment.

We hypothesize:

**H7:** IS professionals in agile software process environments will report higher professional efficacy than those in plan-driven software process environments.

**Job satisfaction**

Locke defines job satisfaction as “a pleasurable or positive emotional state resulting from the appraisal of one’s job or job experience” (1976 p. 1300). Spector (1997) defines job satisfaction as the degree to which people like their jobs. Smith et al. (1969) states that job satisfaction is the feeling that a worker has about his or her job and it has five dimensions: a) satisfaction with the work itself, b) satisfaction with pay, c) satisfaction with promotion prospects, d) satisfaction with supervision, e) satisfaction with co-workers.

Ouchi and Johnson (1978) collected data from two ideal-type representations of organizational control (behavior vs. clan) and found clan control was positively related to the emotional well-being of the employees.

Prior research reported conflicting findings on how new processes impact the job satisfaction of IS professionals. For example, Orlikowski (1993) reported with the introduction of the CASE tools IS professionals in one organization expressed higher job satisfaction while in another organization IS professionals expressed lower job satisfaction. Finlay and Mitchell (1994) reported developers, especially the less experienced ones, viewed the CASE tools as making their job more interesting and more effective.

Specific to our examination of the agile and the plan-driven software development processes, prior studies were generally in favor of agile. For example, Melnik and Maurer (2006) identified a positive correlation between the level of experience with agile methods and the overall job satisfaction. They found twice as many members of agile teams were satisfied with their jobs compared to members of non-agile teams. They noted the ability to influence decisions, the opportunity to work on interesting projects, and the relationships with users were significant factors. Tessem and Maurer (2007) reported agile practices such as feedback, autonomy, and variety led to high job satisfaction in agile teams. Salo and Abrahamsson (2007) shared similar findings - iterative process and immediate feedback on results led to high job satisfaction. Pedrycz et al. (2011) stated a continuous share of the daily working life influenced the programmers’ job satisfaction. Their research investigated the relationship between job satisfaction and pair programming, an agile practice that pushes job sharing to the
extreme. The results suggest pair programming is positively related to job satisfaction.

Based on these empirical evidences and our understanding of the key practices associated with agile and plan-driven software development, we expect job satisfaction to be higher for IS professionals in agile software process environments than those in plan-driven software process environments. We extend the following hypothesis:

\[ H8: \text{IS professionals in agile software process environments will report higher job satisfaction than those in plan-driven software process environments.} \]

METHOD AND EXPECTED CONTRIBUTIONS

We plan to conduct two studies. Study 1 is a quantitative study. It is a survey of software professionals in organizations that adopt different software process models. All scale items will be drawn from existent literature (e.g. Rutner et al., 2008). We will target our data collection on the software professionals in organizations in the Midwest region. We would like to have at least 200 data points.

Study 2 is a qualitative study. It includes a follow-up study where we hold interviews and focus group meetings. We will also conduct content analysis on the comments the respondents provide in their survey responses.

We believe this study can provide several theoretical and practical implications for researchers as well as practitioners. From a theoretical perspective, this study is expected to be one of the first to take control theory as the theoretical lens to examine IS professionals’ job perceptions in agile and plan-driven software process environments. From the practical perspective, a better understanding of the various dimensions and how they differ will allow organizations to design mechanisms to improve employees’ job perceptions.

REFERENCES (PARTIAL LISTING)