Does Software Process Ambidexterity Lead To Better Software Project Performance?

Narayan Ramasubbu, Anandhi Bharadwaj, Giri Tayi

Working paper. Sept, 2001 – Please do not cite without prior permission

Does Software Process Ambidexterity Lead To Better Software Project Performance?

Abstract

Plan-based and agile software development processes seem diametrically opposed in their approaches, with the former emphasizing discipline and control and the latter promoting flexibility and improvisation. Similar tensions in organizational contexts where efficiency versus flexibility considerations simultaneously jostle for management attention has led to the recognition that ambidexterity or the ability to manage seemingly conflicting demands is an important precursor to organizational success. In this study, we extend the idea of ambidexterity to software development processes and empirically examine the performance implications of the ability of software project teams to pursue process designs that simultaneously facilitate both control and flexibility. Utilizing data from a quasi-experiment involving 424 large commercial software projects of a multinational software services firm, we employ a potential outcomes empirical methodology to examine the causal linkage between software process ambidexterity and project performance. Our results show that projects that encountered frequent requirement changes, larger and complex code-bases, new technologies, higher levels of end-user engagements, and smaller, inexperienced teams tend to choose ambidextrous software process designs over a pure plan-based approach. We find that ambidextrous process design positively contributes to better project performance, including on the average about 9% higher productivity, 50% reduction in delivered defects, 12% reduction in internal defects, and 3% improvement in overall profitability. Complementing the archival data analysis with an in-depth qualitative study of the projects pursuing ambidextrous process designs, we enumerate the different mechanisms employed by the project teams to balance control requirements with needs for realizing flexibility. We discuss the implications of our results and elucidate potential pathways to achieve and sustain ambidextrous process designs in software firms.

Key Words: Ambidexterity, Software Process, Software Engineering, Agile Processes, Control, Flexibility, Quality Assurance

Does Software Process Ambidexterity Lead To Better Software Project Performance?

1. Introduction

Even as software systems have become indispensable to almost every aspect of a firm's value chain, developing high quality and cost effective software continues to remain a major challenge for the software industry. The software industry has been described as a hypercompetitive industry (Lee *et al.* 2010) marked by high velocity innovation (Brown and Eisenhardt 1997) and technological change (Schmalensee 2000) where process innovations have been key to success. Tracing the history of software process innovations, Austin and Devin (2009), recount how the industry evolved from *ad hoc* software development (Dijkstra 1972) to *plan-based* approaches that called for an emphasis on planning to anticipate changing conditions and a focus on standards that reduced idiosyncratic interdependence across the different stages of the software development process (Boehm 1988; Cox 1990).

Plan-based approaches to software development have led to improvements in development productivity, quality and maintainability of the developed software (c.f., Harter *et al.* 2000, 2003; Ramasubbu *et al.* 2008); yet they have been criticized for promoting *inflexibility*, or the inability to respond effectively to rapidly changing user requirements especially in more turbulent environments (e.g., Aaen 2003; Highsmith 2002; Maruping *et al.* 2009). Critics of plan-based approaches contend that software development approaches have to remain flexible enough to be able to revisit specifications and refine requirements even in the late stages of development, especially in dynamic environments (Austin and Devin 2009; Harris *et al.* 2009). An alternate process paradigm described as *agile* software development has emerged as a response to these concerns and emphasizes iterative processes and frequent re-planning designed to adjust to unanticipated changes and new requirements (c.f., Beck *et al.* 2001; Boehm and Turner 2003b; Nerur and Balijepally 2007).

Although plan-based and agile approaches seem diametrically opposed in their approach to the design and development of software, with the former emphasizing *discipline* and *control* and the latter promoting *flexibility* and *improvisation*, recent academic research has focused on reconciling the differences and examining the factors involved in choosing an appropriate software process. A key insight that has emerged from research on these two approaches stresses the environmental contingencies that determine the relative efficacies of plan-based versus agile approaches (Boehm and Turner 2003b; Austin and Devin 2009). Empirical research that has examined how firms choose between these two approaches show that software processes that are aligned with business strategies produce better outcomes, such as when firms pursuing differentiation strategies choose agile approaches to software development (Slaughter *et al.* 2006).

While it is important to distinguish the choices and outcomes of plan-based versus agile process designs, a more nuanced approach calls for empirical research into ways of usefully combining plan-based and agile techniques (Austin and Devin 2009; Harris *et al.* 2009). Similar tensions in other organizational contexts where efficiency versus flexibility considerations simultaneously jostle for management attention has led to the recognition that *ambidexterity* or the ability to manage seemingly conflicting demands is an important precursor to organizational success (e.g., Adler *et al.* 1999; Gibson and Birkinshaw 2004; Raisch and Birkinshaw 2008). The idea of ambidexterity in organizations recognizes that demands in task environments are always to some degree in conflict and so there are always trade-offs to be made. Although these trade-offs can never entirely be eliminated, successful organizations reconcile them to a large degree, and in doing so enhance their long-term competitiveness (Adler *et al.* 1999; Raisch *et al.* 2009).

In this study, we extend the idea of ambidexterity to software development processes and empirically examine its impact on software project performance. We define software process ambidexterity as the ability of a software development project team to pursue process designs that simultaneously facilitate both control and flexibility. Understanding the drivers and consequences of software process ambidexterity is an important step towards uncovering mechanisms that shift the key tradeoffs observed in software development such as flexibility *versus* efficiency (Harris *et al.* 2009), productivity *versus* quality (Krishnan *et al.* 2000; MacCormack *et al.* 2003), and development *versus* maintenance performance (Banker *et al.* 1998), which tend to lockdown software firms in vicious cycles of sub-optimal performance.

While the logic of software process ambidexterity has its roots in prior conceptual work bridging the views of plan-based and agile methods (e.g., Paulk 2001; Boehm and Turner 2003a, 2003b; Lee *et al.* 2006, 2007; Vinekar *et al.* 2006), much of the analysis has relied on anecdotal or limited sample case-based evidence, and rigorous large-sample empirical investigation of software process ambidexterity has been scarce. The aim of this study is to fill this critical gap. We examined a large scale field quasi-experiment on software processes at a leading multinational software services company involving 424 commercial software projects, and used a potential outcomes research methodology (c.f., Rosenbaum and Rubin 1983; Mithas and Krishnan 2009) to analyze the observational data for answering the research question: *does software process ambidexterity lead to better software project performance?* Subsequently, we complemented the archival data analysis with an in-depth qualitative study of the projects that pursued ambidextrous software process designs to enumerate the different balancing tactics employed by the teams.

The remainder of the paper is structured as follows: In the next section, we provide the background literature on plan-based and agile software process designs and discuss the antecedents of software process ambidexterity. In section 3, we report the quasi-experiment at our research site and our data collection procedure. In section 4, we present our potential outcomes data analysis and results. Then, in section 5, we enumerate our findings from a qualitative study of the project teams following ambidextrous process designs with a focus on uncovering and inferring the balancing tactics used by the teams. We discuss insights from the results in section 6 and draw implications for an actionable pathway to achieving software process ambidexterity along with acknowledging the limitations of the study and highlighting avenues for future research.

2. Software Process Ambidexterity

We begin with an overview of plan-based and agile software development process designs highlighting the underlying characteristics and the tradeoff challenges that these process designs impose on software development teams. Then, drawing on the organizational ambidexterity literature, we show how these two approaches may be reconciled through ambidextrous software process design and discuss the antecedents of ambidextrous process design.

2.1. Plan-based Software Processes

Grounded in principles of systems engineering and total quality management, a plan-based approach to software development emphasizes *structured processes* throughout the development lifecycle. Detailed plans for the requirements analysis, software design, development, and testing phases are drawn out, and as a project progresses in the development lifecycle adherence to the plans is monitored and documented. Through detailed documentation, plan-based software processes enforce traceability and control of different activities of the project members, inherently disciplining them. Moreover, detailed project metrics are collected at different stages of the project and used for planning and statistical quality control procedures (c.f., Gopal *et al.* 2002; 2005).

The Capability Maturity Model (CMM) (Paulk *et al.* 1993) is a popular and influential software process framework that embodies a typical plan-based process design. The framework was designed to aid firms to improve their software process maturity in planned evolutionary stages (from an ad hoc level-1 to optimized level-5). In a plan-based process improvement paradigm, such as the one advocated by the CMM framework, adherence to the prescribed software processes is periodically audited and assessed by internal and external auditors. To be successfully assessed at a certain capability level, it is typically mandated that a project team need to consistently practice at least 90% of the Key Process Areas (KPAs) pertaining to the chosen capability level, which is governed by the detailed specifications of the plan-

based process framework (e.g., CMM) adopted by the project. Thus, a plan-based software process design enforces a metric-driven and disciplined approach to measure, control, and continuously improve software practices, enabling organization-wide standardization of processes, improving predictability, and reducing uncertainties through improved planning – important tenets of systems engineering (Hall 1962) and total quality management (Hendricks and Singhal 1997).

Plan-based software processes have been widely adopted by software firms worldwide, and there is a continued growth and refinement of the commercially available prescriptive plan-based process frameworks (SEI-PARS 2010). Research shows that adoption of the key processes embodied in the CMM family of process frameworks is associated with positive performance outcomes in co-located (e.g., Krishnan and Kellner 1999; Harter *et al.* 2000; Harter *et al.* 2003) as well as in distributed software development (e.g., Ramasubbu *et al.* 2008; Cataldo and Nambiar 2009).

Despite the many advantages, plan-based processes have been implicated for their *inflexibility* and for giving little room for reflection and improvisation (Aaen 2003; Galliers and Swan 1997; Boehm and Turner 2003b). Aaen (2003) argues that plan-based software process designs have serious shortcomings including those of "substituting technological contact for human interface, gold-plating processes at the expense of knowledge flow, restricting reflective dialogue between participants and leaving no room for experimentation and improvisation" (pp.88). Also, since a plan-based process design attempts to standardize organizational processes, it tends to propagate a blueprint approach to software development, which externalizes process knowledge by formalizing it outside the process user's thought process, and thereby creating artificial structures far removed from actual developers' culture, eventually stymieing improvisation and experimentation (Aaen 2003). These limitations of plan-based process designs hinder a software firm's capability to improvise responses to changing customer needs (due to changes in project requirements, budgets, schedules, etc) and market conditions – essentially losing out on the flexibility dimension of the flexibility-efficiency tradeoff dynamics observed in software development projects (Harris *et al.* 2009).

2.2. Agile Software Processes

Agile software processes emerged as a response to the persistent problem of inflexibility inherent in the plan-based approach to software development. Conboy (2009), offering a robust definition of agile software process, notes that agile processes facilitate "the continual readiness of an ISD [Information Systems Development] method to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value (economy, quality, and simplicity)…" (pp. 340). Despite considerable variation in the agility characterizations among the many commercially available agile software process frameworks (c.f., Boehm and Turner 2003b; Conboy

2009), there is a strong commonality in their adherence to iterative software production and the lean manufacturing principle of "elimination of waste" (Succi 2006).

Through a lean production approach, agile software process designs focus on "individuals and interactions over processes and tools; working software over comprehensive documentation; customer collaboration over contract negotiation; responding to change over following a plan" (Beck *et al.* 2001). Similar to lean manufacturing (Cusumano 1994), the agile software process design allows independent programmers led by trustworthy project managers to respond to changing customer needs by making rapid and quick changes to any aspect of a software project without facing the bureaucratic hurdles of approvals and detailed justifications that are typical of plan-based software processes. In contrast to plan-based software processes, agile processes encourage social inquiry and collective action, focusing on improvisation and rapid adaptations to respond to changes. Agile software processes tend to align more towards the flexibility dimension in the flexibility-efficiency tradeoff dynamics seen in software development (Harris *et al.* 2009). For example, a large-scale study at Microsoft reported design flexibility and the ability to quickly respond to requirements changes as one of the top benefits realized due to the adoption of agile software processes (Begel and Nagappan 2007). Dyba and Dingsoyr (2008) summarize the findings of more than thirty empirical studies that reported positive impacts of agile software process design on a variety of performance outcomes including improved relationships with customers, the ability to incorporate requirement changes even at later stages of a project, improvements in team cohesion and programmer job satisfaction, and significant improvement in product quality.

However, agile software process designs are not without limitations. Doubts have been raised over the ability of agile software process designs to scale for large software projects, especially those distributed across different time zones (Turk *et al.* 2002; Lee *et al.* 2006; Begel and Nagappan 2007). Some studies have reported a negative impact of agile software processes on productivity, mostly citing work pattern disruptions and other inefficiencies induced due to the frequent informal meetings encouraged by agile methods (Begel and Nagappan 2007; Dyba and Dingsoyr 2008). Parnas (2006) highlights the dangers of a "no document lean process" culture propagated by agile software process designs and notes that it is risky to rely on oral interactions to communicate detailed and precise software requirements. Similarly, Lee *et al.* (2006) note that conventional agile methods need to be modified to embrace more rigor and discipline in order to overcome communication and knowledge sharing challenges in globally distributed software development. Other scholars have highlighted a variety of organizational-, people-, and technology- related challenges of adopting agile software processes, especially for firms new to an iterative, collaboration-centric software development paradigm (Nerur *et al.* 2005).

2.3. Ambidexterity: Balancing Discipline and Agility

Plan-based and agile software processes represent opposite ends of a process design continuum emphasizing efficiency and control on the one side and flexibility and responsiveness to change at the other end. Given these extreme ends at the software process design continuum, process designs that are capable of balancing the tradeoff at different stages of a software project development cycle, rather than the choices at the extremes, can be expected to yield superior performance outcomes. For example, Adler *et al.* (1999) reported how a Toyota subsidiary achieved superior flexibility-efficiency combinations through dynamic adjustments of the routine (efficiency-focused) and non-routine (experimentation-focused) components of its organizational processes during turbulent periods of model changeovers. Organizational ambidexterity, i.e., the ability of firms to be responsive to changes while at the same time being able to carry out current activities efficiently, has a rich research tradition spanning several disciplines (e.g., Duncan 1976; Gibson and Birkinshaw 2004; O'Reilly and Tushman 2004; Raisch and Birkinshaw 2008). A key postulation of the theoretical view is that ambidextrous firms institute process designs that dynamically combine exploitation (standardized, efficiency-focused routines) and experimentation (improvised, flexible, agility-focused routines) activities in order to shift challenging tradeoffs and achieve superior performance outcomes. In the Information Systems Development (ISD) literature, several theoretical conceptualizations of ambidextrous software process designs that support both flexibility and control have been proposed (e.g., Vinekar *et al.* 2006; Lee *et al.* 2007). Drawing on tenets from the theory of dynamic capabilities and control theory, Harris *et al.* (2009) propose controlled-flexible process designs that adopt "emergent outcome controls" as a way to avoid the sub-optimal process choices at the extremes of the process design continuum.

Broadly, three key categories of antecedents for ambidextrous process capabilities have been identified in the organizational literature, namely, *structural-*, *contextual-*, and *leadership-* based antecedents (Raisch and Birkinshaw 2008). Structural antecedents, as the name implies, refers to the structural mechanisms that are put in place to deal with important tradeoffs that organizations face. Such mechanisms include, for example, spatial separation with separate units in charge of plan-based and agile processes and temporal partitioning with the same unit using both plan-based and agile process at different points in time. Contextual antecedents refer to the systems, processes, and beliefs that a firm puts in place to encourage individuals to handle the conflicting demands on their time in a desired way. In the software development context, this includes, for example, coping mechanisms for individuals dealing with the conflicting demands of documentation versus experimenting with new technologies (Lee *et al.* 2006). Finally, the role of leadership, including the support of senior executives has been identified as an important antecedent of organizational ambidexterity.

In the ISD literature, Vidgen and Wang (2009), propose three principles rooted in complex adaptive systems theory to design ambidextrous software processes that can lead to better performance: matching or exceeding rates of change in the environment, creating necessary conditions for self-organization, and creating routines for synchronizing exploration and exploitation activities. Austin and Devin (2009) developed a contingency framework that establishes the key conditions for the viability of ambidextrous process designs that realize efficiencies without sacrificing flexibility. The framework posits that an ambidextrous process design is viable only when it leads to a reduction in reconfiguration, coordination, and context-dependent experimentation costs leading to surpluses in software production. Based on their examination of 22 globally distributed software projects, Lee *et al.* (2006) propose a set of coping strategies that promote both flexibility and rigor in global software development and help teams respond to environmental changes efficiently. Boehm and Turner (2003b) propose a framework consisting of several software project-specific contextual variables such as the capability of the development team, customer involvement, requirements volatility, project size, and code complexity to balance agility and discipline in process designs.

In summary, much of the theoretical work on ambidextrous software process design takes a distinctly prescriptive approach, laying out the conditions and parameters of ambidextrous design choices, with the underlying and untested assumption that such ambidexterity leads to better software project performance. We set out to explicitly test this assumption in the current study. With the goal of establishing a causal linkage between ambidextrous software process design and software project performance, we conducted an empirical investigation using a quasi-experimental setup.

3. Ambidextrous Process Design: Quasi-Experiment at Research Site

Our research site is a leading multi-national software development firm operating in 55 countries with over a hundred thousand employees and more than 6 billion dollars in revenues in 2010. A majority of the software development centers operated by the firm were assessed at CMMI level-5[1] and the firm was also a recipient of the IEEE Software Process Achievement award (IEEE SPA 2010). The firm had a centralized Software Engineering Process Group (SEPG) that was responsible for governance of development processes. The SEPG had invested heavily in standardizing the development processes prevalent in the firm through the rigorous deployment of a plan-based process paradigm using the CMMI process framework. While project-specific tailoring of the CMMI Key Process Areas (KPAs) was allowed, the tailoring of processes and usage of any non-standard processes employed by individual

[1] CMMI is an integrated process improvement framework developed by the Software Engineering Institute at the Carnegie Mellon University. Level-5 of the CMMI is the highest maturity level indicating quantitatively controlled and well-optimized processes.

projects typically needed prior approval and were actively monitored by the SEPG personnel. All project teams included dedicated Software Quality Assurance (SQA) personnel who were independent project team members reporting directly to the SEPG and not to the respective project managers.

While the plan-based process paradigm implemented using the CMMI family of process frameworks had served the firm well, the SEPG wanted to adequately prepare itself for a turbulent environment of software process diversity as the firm embarked on a period of rapid business growth through ambitious acquisitions and expansions to new markets. Senior executive management enthusiastically supported and worked with the SEPG towards the launch of an organization-wide initiative to introduce a "controlled-flexible" process design that fused agile software processes with the existing CMMI plan-based process framework. Separate financial resources were allocated for the initiative that covered training and other miscellaneous needs. A chosen SEPG team championed the initiative on a full-time basis and no other production duties (i.e., activities concerning a live commercial project) were assigned to this team. Thus, the structural, contextual, and leadership based process capabilities, which serve as important antecedents of organizational ambidexterity (Raisch and Birkinshaw 2008), were adequately fulfilled in the context of our research site.

To develop the new controlled-flexible process design, the SEPG teams mapped the extensive CMMI KPAs with the chosen agile process frameworks (Extreme Programming (XP) and SCRUM) for operations under a seamless and uniform process governance framework[2]. The SEPG also implemented a set of collaboration and agile-methods specific tools for aiding projects to collect appropriate metrics and integrate them with the central project and process database of the firm. It is noteworthy to mention that while the SEPG had allowed more diversity of project-level software processes, no extensive changes were made to the central process governance structures, including the presence of independent SQA personnel in projects. Thus, the firm had put in place the adequate control mechanisms necessary to govern the introduction of new agile process components into established organizational routines.

The SEPG began to roll out the newly developed "controlled-flexible" process designs to newly starting projects by inviting "heavyweight" project managers to participate in a pilot roll out. During the pilot roll out, project managers and team members who volunteered to use the new "controlled-flexible" process design in their projects were first provided with adequate training in agile software process methods (XP and SCRUM) and the process mapping framework before they proceeded to tailor their project-level production processes. The initial success of the pilot projects and the positive perceptions of the participating project teams provided the necessary impetus for a broad roll out of the ambidextrous

[2] A generalized example of such a mapping of CMM KPAs and Extreme Programming framework can be found in Paulk 2001.

"controlled-flexible" process design to all worldwide development centers of the firm. The choice of process design options (and several training modes) available for project teams was widely advertised throughout the firm. Project managers and team members of all newly starting projects had the full autonomy to choose between the new ambidextrous "controlled-flexible" process design and the standardized (and well-established) plan-based process design depending on their own project contexts. At the time of our data collection, 154 projects had adopted an ambidextrous process design instead of a pure plan-based approach.

3.1.1. Data Collection from the Quasi-Experiment

We collected data from our research site in three phases. In the first phase, we conducted multiple field observations spanning over a month when one of the authors was resident at the research site. We interviewed the executive management and SEPG of the firm to understand the organizational context of the software projects-level data that we had planned to collect. Table 1 maps the key antecedents discussed in the organizational ambidexterity and process literature with our firm-level observations from the first phase of data collection. As mentioned in the previous section, our observations presented in Table 1 confirms that the empirical context exhibited the important structural-, contextual-, and leadership-related antecedents of ambidexterity identified in the literature, and that the quasi-experiment at the research site provided an excellent context to examine the effects of software process ambidexterity on project performance outcomes.

In the second stage of our data collection, we utilized the software process selection and comparison framework proposed by Boehm and Turner (2003a, 2003b, pp.25-58) as a conceptual foundation to collect project-level data. The framework posits consideration of quantitative metrics such as the capability of the development team, customer involvement, requirements volatility, project size, and code complexity to assess the specific contextual requirements of a software project team in order to derive hybrid designs that balance agility and discipline. The second stage of our data collection effort resulted in an archival data set from 424 commercial software projects completed by the firm. The 424 projects were the sample used in a recent CMMI capability reassessment exercise by an external team of CMMI auditors from KPMG. Since these projects had been audited multiple times by the external KPMG auditors and the firm's SEPG team, high confidence can be placed on the reliability and accuracy of the data. The variables in our data set gathered in the second stage of our data collection are presented in Table 2.

In the third stage of our data collection, we embarked on a qualitative study through in-depth discussions with the project managers, team leaders, and team members of 154 software projects that implemented ambidextrous software process designs over a six-month period. The qualitative study

complements the large scale econometric analysis of the archival data collected in the second stage. The main goal of the qualitative study was to uncover in richer detail the individual balancing strategies that the projects teams employed in their ambidextrous process designs.

Table 1. Summary of Firm-level Observations

	Literature Reference	**Observation at our research site**
Ambidextrous Process Design Decision Factors		
Ambidexterity benefits	Raisch and Birkinshaw (2008); Austin and Devin (2009)	• Ability to accommodate process diversity due to mergers and acquisitions. • Rapid business growth and survival in a hyper-competitive industry.
Ambidexterity costs		• Increase in risks due to loss of predictability and control. • Additional governance expenditures.
Organizational Antecedents of Ambidexterity		
Structure	Gibson and Birkinshaw (2004); Raisch and Birkinshaw (2008)	• Independence and autonomy of SEPG personnel and Project Management personnel.
Context		• Adequate support for project personnel in designing, tailoring, processes and reporting metrics. • Voluntary participation in experimentation.
Leadership		• Executive management support. • Adequate financial resources for process design experiment.
Governance and Performance of Ambidextrous Process Design		
Emergent Outcome Controls	Harris *et al.* (2009); Maruping *et al.* (2009)	• Incremental metrics collection for different iterations. • Facility for both plan-based and improvised and collective decision making (through collaborative tools).
Common Platform	Lee et al (2006, 2007, 2009); Vinekar *et al.* (2008)	• Common SEPG governance of all process designs (through common mapping of KPAs and metrics). • Technology readiness through appropriate collaborative tools.
Project Context and Performance	Boehm and Turner (2003b)	• Detailed project-level metrics collected for statistical quality control purposes. • Refer to Table 2 for list of variables.

It is important to note that we did not experimentally control the allocation of the different software process designs for the projects, but only observe the choices made by the software teams and the corresponding performance consequence of those choices. In the absence of random assignment, observational data from the quasi-experiment encounters selection problems, and therefore causal interpretation by comparing the performance outcomes of groups following two different process designs becomes problematic. Potential outcomes methodology overcomes this difficulty by viewing causal effects as a comparison between two potential outcomes at a given time corresponding to a treatment that was applied, and addresses the selection bias problem through propensity score matching techniques

(Rosenbaum and Rubin 1983; Dehejia and Wahba 2002). Potential outcomes research method has been widely utilized in statistics, economics, and sociology to examine issues of causality, similar to our research question, through observational data (c.f. Rubin 2005). More recently, Mithas and Krishnan (2009) articulate the use of the methodology to answer research questions related to the information systems domain. The potential outcomes research methodology that we employ to analyze our quasi-experimental data is explained in the next section and closely resembles the approach outlined by Mithas and Krishnan (2009).

Table 2. Archival Data Variable Definitions

Variable	Definition
Software process ambidexterity	This is the treatment variable in the quasi-experiment; coded as 1 if projects used the "controlled-flexible" process design; coded as 0 if projects used the CMMI plan-based process design
Productivity	Function Points / Total project effort
In-process defects	Count of defects logged in the project before project delivery to the customer
Delivered defects	Count of defects logged in the project after project delivery to the customer (during the warranty period)
Profitability	% Profits / Total cost of the project
Requirements volatility	% Total effort spent on rework due to change in customer requirements
Newness	Dummy variable coded as 1 if the technology or/and design involved in the project was new to the project team (no prior experience). The value was self-reported by project managers.
Client involvement	% Total effort spent on engaging with end users
Reuse	% Code that was reused from existing libraries (either at the firm or from the customer)
Team size	Full time headcount of personnel involved in the project
Team experience	Average professional work experience of project team (in years)
Project size	Forward counted function points
Project manager certification	Dummy variable coded as 1 if the project manager possessed any professional certifications (e.g., PMI, SCRUM professional, etc); 0 otherwise
Contract	Dummy variable coded as 1 if the project followed a fixed price contracting scheme; 0 for time and materials

4. Archival Data Analysis

4.1. Average Treatment Effect for Ambidextrous Process Design

The first step in implementing a potential outcomes-based analysis through propensity score matching is to identify the treatment, outcomes of interest, and other covariates. In this study we define the choice of ambidextrous process design as the treatment, project performance variables as outcomes,

and the several ambidexterity antecedents identified through prior research as covariates. The causal effect we are interested in analyzing is the performance outcomes for the projects that chose an ambidextrous process design for software production. Since the covariates we selected for analysis were derived from a variety of well-established theories spanning the organizational ambidexterity, control theory, complex-adaptive systems, and software engineering literatures, we are confident in making the strong ignorability assumption involved in propensity score matching (Rosenbaum and Rubin 1983; Rosenbaum 1984) – i.e., we assume that the selection bias because of the lack of random treatment is mostly due to the correlation between the observed covariates and process design, and not because of other unobserved mechanisms. In section 4.3, we provide a sensitivity analysis to estimate the extent to which our study may be vulnerable to what we may have missed. Table 3 compares the observed characteristics of projects that chose an ambidextrous process design with the projects that chose the regular plan-based process of the firm.

Table 3. Characteristics of Treatment and Control Groups Before Matching

Variable	Treated Sample (Ambidextrous process)	Control Sample (Plan-based process)
Sample size N	154	270
Requirements volatility	20.27***	12.83
Newness	0.543***	0.233
Client involvement	22.214***	14.18
Reuse	4.577***	2.007
Team size	9.5143	11.407*
Team experience	3.898	3.7445
Project size	1757.5*	1291
Project manager certification	0.521**	0.393
Contract	0.521	0.585

Note: Significance levels for differences in means using t-tests on the larger of the two numbers across treatment and control groups; *p <0.10;**p<0.05;***p<0.01. F-statistic of Hotelling's T-squared test for all covariates was significant at P<0.01.

From Table 3, we can see that there are statistically significant differences across the variables in the treated and control samples, lending support to the view that the choice of ambidextrous process design by certain projects was likely a non-random choice. On average, as highlighted in prior conceptual literature (Boehm and Turner 2003b; Vinekar *et al.* 2008; Austin and Devin 2009) projects that encountered higher level of requirements changes, end user engagements, and worked with newer technologies preferred ambidextrous processes over pure plan-based process designs. Also, smaller teams and projects that worked on larger code-bases and those with project managers who possessed professional certifications chose the ambidextrous process design. Note that the treatment and control groups have not yet been matched using propensity scores to account for the non-random assignment.

In order to adjust for the non-random assignment in the quasi-experiment data, we calculated the propensity scores (i.e., the propensity of a project to use ambidextrous process design) using a logit model and employed a kernel-matching estimator. Our specification of the logit model was informed by prior theory regarding the antecedents of ambidexterity (section 2.3) and is given in equation 1.

$$\text{logit(ambidextrous process choice)} = \beta_0 + \beta_1*(\text{Requirements volatility}) + \beta_2*(\text{Newness}) + \beta_3*(\text{Client involvement}) + \beta_4*(\text{Reuse}) + \beta_5*(\text{Team size}) + \beta_6*(\text{Team experience}) + \beta_7*(\text{Project size}) + \beta_8*(\text{Project manager certification}) + \beta_9*(\text{Requirements volatility}) + \beta_{10}*(\text{Contract}) + \varepsilon \quad \text{.....Eq(1)}$$

Table 4. Logit Parameter Estimates for Propensity Score Calculation

Variable	Ambidexterity (logit model)
Requirements volatility	0.016*** (0.007)
Newness	1.629*** (0.00)
Client involvement	0.0745***(0.000)
Reuse	0.132*** (0.004)
Team size	-0.057*** (0.004)
Team experience	-0.139** (0.019)
Project size	0.192*** (0.002)
Project manager certification	0.382 (0.145)
Contract	0.231 (0.387)
Chi-Square	137.55*** (0.00)
Log-likelihood	-202.919
Goodness of fit	AIC = 1.004, BIC = -2098.752

Note: The model included an intercept; robust p-values in parenthesis; *p <0.10;**p<0.05;***p<0.01

The logit parameter estimates are presented in Table 4. The Chi-squared test of the logit model shows that the selection model is significant compared with a model with no explanatory variables. Thus, we can conclude that the projects that chose ambidextrous process design differ significantly from those that chose plan-based process design with respect to the observed covariates. As posited by prior conceptual studies, projects that encountered frequent requirement changes, new technologies, higher levels of end user engagements, and smaller teams tend to choose ambidextrous software process designs over a pure plan-based approach. In contrast to expositions of prior research, we observe that experienced personnel at our research site preferred plan-based approaches and projects that handled larger (and more complex) code-bases chose ambidextrous process designs, suggesting the presence of both uncertainty-avoiding and risk-taking behavior profiles among project teams at the research site.

When we employed kernel matching to calculate the propensity scores, matching estimators could not identify treatment effects for four observations that did not fall in the region of common support

(between propensity score estimations and kernel matching estimations). Bias stemming from non-overlapping support is typically attributed to selection biases, and since we lost only four observations, it is not a signficant concern for our analysis. Table 5 lists the characteristics of the treatment and control groups after matching. As compared to the disparities between the treatment and control groups before matching (Table 3), we can see that the matched treated and control sample are very close to each other with respect to the observed covariates. Thus, a considerable amount of bias induced due to non-random assignment in the quasi-experiment has been reduced through propensity score matching.

Table 5. Characteristics of Treatment and Control Groups After Matching

Variable	Treated Sample (Ambidextrous process)	Control Sample (Plan-based process)
Sample size N	154	270
Requirements volatility	20.025	18.681
Newness	0.537	0.558
Client involvement	21.966	22.75
Reuse	4.1881	3.9969
Team size	9.625	10.53
Team experience	3.8	4.1
Project size	1800	2462.6
Project manager certification	0.507	0.539
Contract	0.529	0.542

Note: t-tests for differences in means were performed and no differences were found even at $p \leq 0.10$, suggesting a good matching on all observed covariates. We also conducted Hotelling's t-squared test for all covariates. The resulting F statistic was not significant.

In order to assess the performance implications of ambidextrous process designs and establish a causal linkage between process choice and several dimensions of project performance, we computed the average treatment effect of ambidextrous process design. Using the propensity scores, we utilized the Gaussian function for Kernel matching, to calculate average treatment effect on the treated sample across productivity, delivered defects, in-process defects, and overall project profitability variables. These results are presented in Table 6. We find that the ambidextrous process design positively contributes to achieving better project performance, including on the average about 9% higher productivity, 50% reduction in delivered defects, 12% reduction in internal defects, and about 3% improvement in overall profitability. This result establishes the causal linkage between software process ambidexterity and better project performance and answers the key research question of the study. While we have empirically established that ambidextrous process designs improve performance, given the turbulent project environments and varying contextual factors, it is not clear if all projects benefit equally from process ambidexterity. We assess this question in the next section by analyzing treatment effect heterogeneity.

Table 6. Overall Treatment Effect on Treated Using Kernel Matching

	Treated	Control	Difference
Productivity			
Before Matching	0.266	0.132	0.133
After Matching*	0.267	0.243	0.023** (9.46%)
Delivered defects			
Before Matching	28.293	27.963	0.329
After Matching*	29.1	58.244	-29.148** (-50.04%)
In-process defects			
Before Matching	517.979	463.144	54.834
After Matching*	509.081	580.424	-71.343** (-12.29%)
Profitability			
Before Matching	44.477	39.243	5.234
After Matching*	44.413	41.678	2.734** (%)

Note: *Kernel Matching using Gaussian function; **Average treatment effect on the treated

4.2. Treatment Effect Heterogeneity

In this section, we analyze the heterogeneity of the treatment effect to examine whether all software projects benefited equally from an ambidextrous software process design. Following guidelines from prior research (Dehejia and Wahba 2002; Mithas and Krishnan 2009), we used propensity score stratification and classified all observations in to five subclasses based on estimated propensity scores. We made sure that the covariates balanced across treatment and control units in each of the five strata enabling a fair comparison of the treated and control groups. Figure 1 shows the distribution of propensity scores of treated and control subjects in each stratum. Based on the propensity scores, projects in stratum 1 were predicted to have the lowest propensity to adopt an ambidextrous process design whereas projects in stratum 5 had the highest propensity to use ambidextrous process design for their operation. Figure 2 shows the variation of the estimated average treatment effect, i.e., the effect of choosing an ambidextrous process design on the four performance variables that are used in this study within each stratum of projects.

The results from our treatment heterogeneity analysis imply that projects that were least likely to adopt an ambidextrous process choice benefit more if they chose an ambidextrous process design. For example, adoption of the ambidextrous process design by the projects in stratum 1 would result in about 141% increase in productivity, 71% reduction in delivered defects, 17% reduction in internal defects, 43% improvement in profitability – overall a significant improvement in project performance. On average, the projects in stratum 1 (low propensity to adopt ambidextrous processes) had personnel with more professional experience. Our results show that if these highly experienced personnel were provided

with agile process methodology training and encouraged to balance their chosen plan-based approach with elements of agile methodologies as mapped by the SEPG, significant performance benefits could be realized.

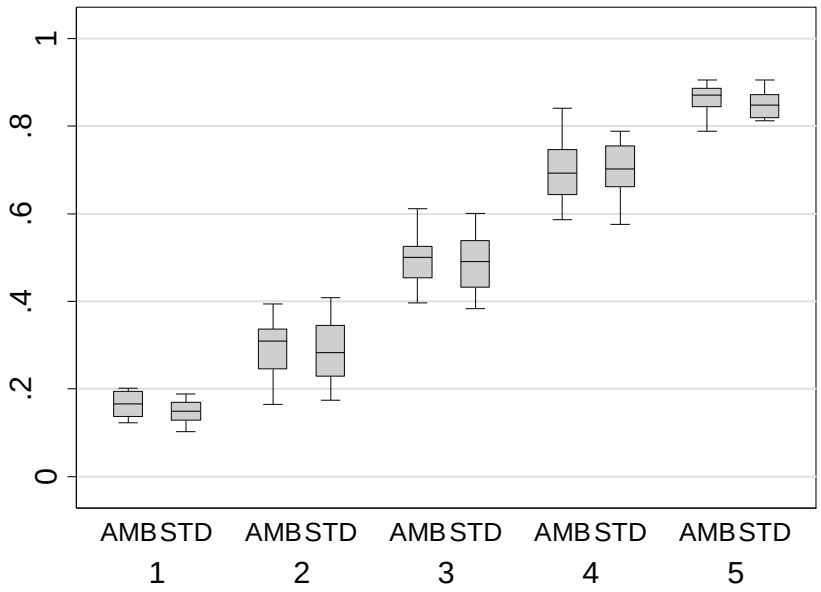

Figure 1. Propensity Score Distribution Across Strata
Note: AMB: ambidextrous process design; STD: standard plan-based process design

Projects in stratum 5 (projects that were more likely to adopt an ambidextrous process) exhibit an interesting behavior. On average, they perform poorly with respect to productivity (33% lower) but significantly outperform with respect to decreasing delivered defects (83% lower defects). A significant chunk of the projects in this stratum were very large projects (with respect to code-base size) that chose to adopt the ambidextrous process design. These projects seem to depict the classic productivity-quality tradeoff encountered in large software products (Krishnan *et al.* 2000) – they seem to focus on improving quality at the expense of productivity. Despite the presence of such a tradeoff, through an ambidextrous design of their process, these projects are still able to post better overall performance with respect to profitability (12% higher).

In sum, our analysis of the treatment effect heterogeneity shows that not all software projects benefit equally along all the four project performance outcomes by adopting an ambidextrous software process design. We discuss the implication of this result for the design of ambidextrous processes in section 6.

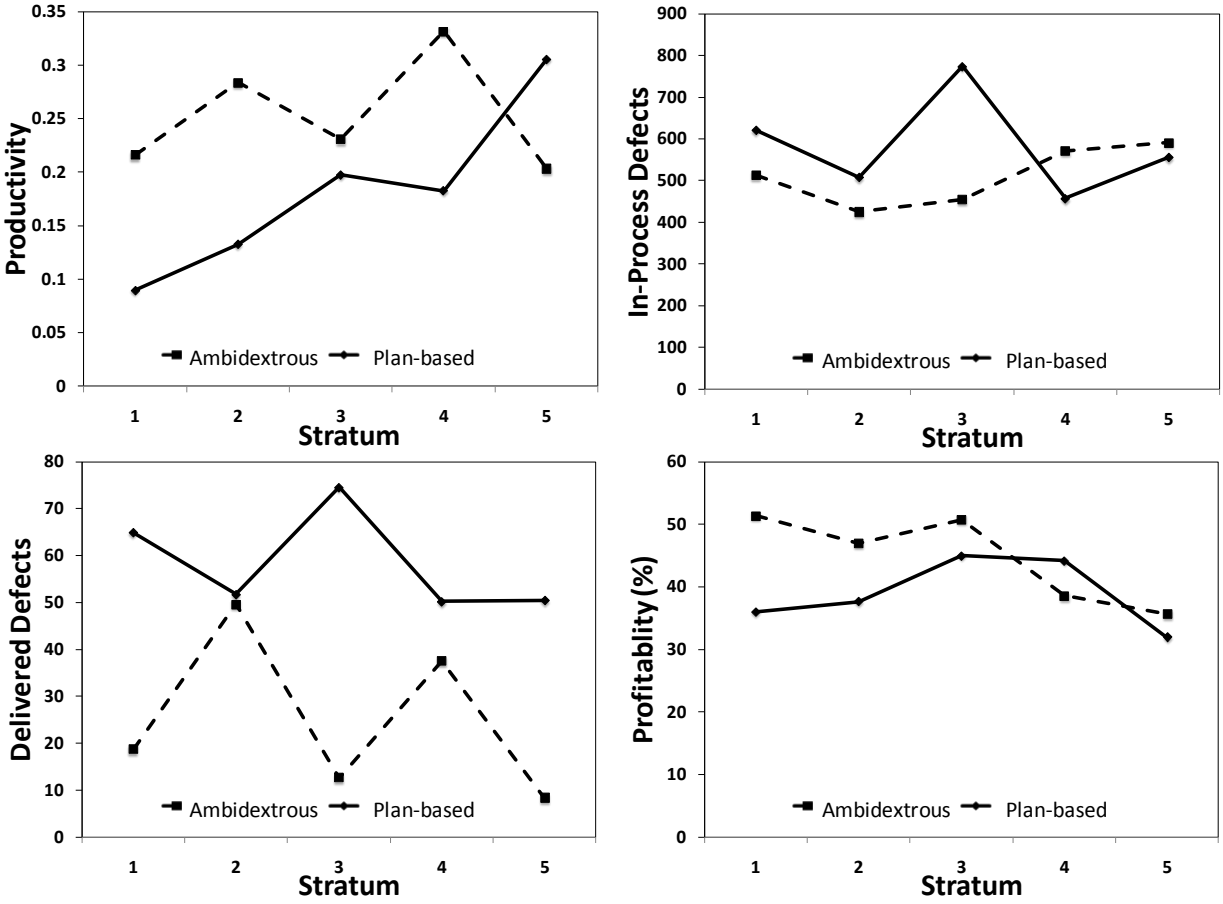

Figure 2. Treatment Effect Heterogeneity

4.3. Sensitivity Analysis

We analyze the sensitivity of estimated causal effects presented in Table 6 to potential violations of the strong ignorability assumption of the propensity score methodology (Rosenbaum 1984). These results are presented in Table 7. The sensitivity analysis tests the extent to which our results from the propensity score analysis is robust to any potential unobserved characteristics that we did not account for in the ambidextrous process choice model (logit model specified in equation 1). Gamma reported in Table 7 measures the hypothetically induced differences to treatment and control group assignments as a result of potential unobserved characteristics. For example, Gamma=5 when compared to Gamma=1 signifies a 500% difference in treatment and control group assignments (from the one used by this study) due to potential unobserved characteristics. Our results reported in Table 7 show that even at a very large hypothetically induced bias (differences to treatment and control group assignments), the causal effects predicted by our analysis are robust. Thus, high confidence can be placed on our results and the theoretically guided choice of covariates used for potential outcomes analysis of this study.

Table 7. Sensitivity Analysis of Ambidextrous Process Design

Gamma*	Significance Level
Productivity	
1	0
5	0
5.5	$1.2e^{-15}$
6	$1.2e^{-14}$
Delivered Defects	
1	0
5	0
5.5	$2.2e^{-16}$
6	$3.2e^{-15}$
In-Process Defects	
1	0
5	0
5.5	$5.6e^{-16}$
6	$8.8e^{-15}$
Profitability	
1	0
5	0
4.5	$1.1.e^{-16}$
5	$5.9e^{-15}$

Note: Sensitivity analysis on the average treatment effect on the treated; *Log odds of differential assignment to treatment because of unobserved factors

5. Qualitative Study

The main goal of our qualitative study was to uncover patterns on how the project teams went about balancing the plan-based and agile components in their ambidextrous software process designs in order to achieve superior project performance. The qualitative study involved discussions with project managers, module leaders, and team members of the 154 software projects that followed ambidextrous process design for software development (i.e., the treated sample in the quasi-experiment). We held a total of thirty group discussion sessions in three rounds over a six-month period. Each discussion session had at least one project manager, module or team leader, and programmer and lasted about an hour. The first round of discussions were open-ended and focused broadly on describing the projects, client relationships and behavior, work breakdown structures, specific challenges encountered in the projects, technologies involved, project performance, and unique software processes followed in the projects including what the participants considered as best practices. Similar to other qualitative software process studies (Sarker and Sarker 2009), we analyzed the data from our discussions in an iterative fashion using constant comparisons and allowing for inductive reasoning.

Our analysis of the first round of discussions corroborated the broad patterns observed in the archival data analysis, but also raised several questions on how the teams went about balancing the plan-based and agile components. We therefore conducted two more rounds of discussions with the same set of individuals. The discussions in the last two rounds of the qualitative study were more narrowly focused

on the specific dimensions of the balancing strategies identified in the initial round. Our notes from the three rounds of discussions form the basis of our qualitative analysis and inference.

We began coding the interview transcripts using *in vivo* codes using the lexicon of terms discussed by the respondents. As we coded the data, we looked for emergent themes, visualizing the relationship between different parts of the data and established theoretical ideas in the literature. The main theoretical concepts that guided our analysis, comparison, and inference were related to the tradeoff shifting mechanisms reported in prior organizational ambidexterity literature (e.g., Adler *et al.* 1999; Gibson and Birkinshaw 2004) and the emerging body of literature on controls in agile and flexible software development (Boehm and Turner 2003b; Harris *et al.* 2009, Maruping *et al.* 2009, Vidgen and Wang 2009). As we progressed through the iterative analysis, we added additional code labels to the transcript passages representing appropriate theoretical constructs that we identified in the ambidexterity and software process literature. Coding reliability was mainly established through two stages of review by an independent academic researcher and a practitioner expert with more than fifteen years of software project management experience. Together, the independent reviewers covered more than 80% of the qualitative data and after resolving a few minor discrepancies, achieved a high level of congruence in accepting our final coding scheme. The inter-rater reliability as measured by Cohen's Kappa was 0.9. We report our findings from the qualitative study in the next section.

5.1. Qualitative Study Results

First, we observed that the balancing tactics employed by project teams spanned multiple dimensions including individuals, teams, software artifacts, and cross-functional processes lending support to prior research propositions that flexible process designs need to be viewed as a multifaceted and multilevel concept spanning people-, technology-, and environment-related factors (Boehm and Turner 2003b; Sarker and Sarker 2009). Further, early in our discussions with the participants, we discovered that the teams often associated their balancing tactics with anticipated or observed changes in their operating environment. In other words, the trigger for a rebalancing act that altered the composition of plan-based or agile elements in a project's process design was often attributed to 'proaction' in advance of change, 'reaction' to change, or 'learning' from change – the primary dimensions identified in the ISD literature to assess process design agility (c.f., Conboy 2009). In the later rounds of our discussions, we utilized this taxonomy to garner specific examples of balancing plan-based and agile aspects in process designs. Overall, the balancing tactics we uncovered can be organized along four key dimensions: *tradeoff shifting mechanisms*, *client relationships*, *artifact specification*, and *project governance*. The specific balancing tactics we uncovered along these dimensions, the corresponding literature references, and the dominant plan-based and agile components of the tactics are summarized in Table 8.

Tradeoff Shifting Mechanisms

Meta-Routines: Discussion participants emphasized that a key enabler of software process ambidexterity in their projects was the extensive use of detailed and standardized process templates, which enabled software development meta-routines regardless of whether the teams used specific plan-based or agile process components in their methods. Meta-routines enabled by the process maps systematize problem-solving procedures (c.f., Adler *et al.* 1999) providing a common organization-wide platform for accommodating changes (Feldman and Pentland 2003). Process maps helped the project teams to be flexible with the specific implementation approach of KPAs of diverse process methods, and simultaneously facilitated independent audit and verification by authorities outside the project team such as the organizational SEPG and external auditors. Moreover, the operational infrastructure for using and reporting process templates and tailoring project-level process designs was fully automated, thus avoiding unnecessary resource overheads for manual reporting and documentation, facilitating lean and well-controlled meta-routines.

Partitioning: Project teams followed context-specific partitioning mechanisms for bounding the scope of plan-based and agile components of their processes. Spatial partitioning of sub-units and temporal partitioning of tasks within a unit have been reported to aid ambidexterity in prior literature (Adler *et al*. 1999; Puranam *et al.* 2006). We did not find any sub-unit structural partitioning mechanisms at our research site, but teams followed temporal task partitioning to balance plan-based and agile process needs. Broadly, when requirements were ambiguous and uncertain, teams started on an agile process footing and moved on to more plan-based methods at later stages of the project when a stable design had evolved. The following narrative from a project manager illustrates a temporal partitioning strategy:

> "A client power-user worked intensely with us for a month at the start of the project. The focus at that time was on rapidly figuring out what the end user wanted, what could be done with the existing systems, and what we could deliver within the schedule. Since no one had answers to all the questions, we had to progress in rapid cycles. Our design kept changing everyday and whiteboard pictures captured on our cell phone cameras was the basis of our documentation. But, once the system design was finalized, we started planning in detail for the rest of the project duration [6 months]. There was no spiraling after that…"

Teams that had started their project on a stable and well-planned footing rebalanced their methods to include more agile components at times of turbulence due to external shocks, as recounted by a team leader:

> "As the project was for a repeat customer, we knew the system well. We made detailed project plans and design document that were approved by the client team. We used to send our [balanced] scorecard reports to the client every week as the build progressed. But, at one point of the project when the client applied a new patch from another platform vendor, an important module stopped working. Suddenly, nothing seemed to progress. Numbers [on the balanced scored report] looked

> so bad that we stopped sending it to the client. We had to suddenly find alternate ways of doing things. That's when we started resorting to problem frames, small releases, continuous integration, and collective ownership of system environment [with the client]. It turned out that the only plan we kept was the delivery deadline; everything else changed on a day-to-day basis."

While the application of specific partition mechanisms varied across the project contexts, what was common among the projects was the use of scope and temporal boundaries for bounding the application of plan-based and agile process components. Although the teams frequently switched between the boundaries of plan-based and agile process components depending on their context, the common platform of process maps, which connected the context-specific process elements to meta-routines, facilitated task-level accounting and traceability of expenditure. The teams handled potential impediments to learning due to the partitioning of plan-based and agile process boundaries and frequent switching between them through a combination of formal and 'on-the-job' training through peer mentors. Learning from each other through peer-reviews and pair programming was the most common informal mechanisms that came up in our discussions. Formal training was handled through an organization-wide education unit that conducted both in-class room and online courses on a variety of project management, quality management, and technical topics.

Client Relationship

Relational capital with clients played an important role in how the project teams balanced plan-based and agile components in the process design. Teams followed user-centered design principles and standards (ISO9241-210, 2010) and had personnel in multiple roles interact with end users and client managers. However, project teams often encountered significant variation in client involvement during the course of a project lifecycle, and the extent of client involvement during specific stages of the project lifecycle determined if collective ownership to issues could be forged or if relationships centered on more formal contractual norms such as service level agreements (SLAs). The following quote from a software engineer captures the logic of balancing agile and plan-based process components in day-to-day work using the client relationship axis:

> "I have to work with different departments of a client. Some [end-users] require detailed documentation and cost justification for everything, others are more trusting and approve changes over a phone call, and a few won't even open documents or read emails [that I send them]. You can't expect all end-users to know what is permitted and what is not permitted according to the project contract, and you can't throw around contract terms to end users who are normally helpful…"

Table 8. Control and Flexibility Balancing Mechanisms

Balancing Dimension	Sample Literature Reference	Balancing Tactic	Discipline and Control Component	Flexibility and Improvisation component
Tradeoff Shifting Mechanisms				
Meta-routines	Adler *et al.* (1999); Gibson and Birkinshaw (2004); Raish and Birkinshaw (2008); Puranam *et al.* (2006).	Use of standardized process mapping template.	Standard mapping of templates facilitate independent auditing and traceability.	Malleability of process components depending on project context.
Partitioning		Separation of process component based on software lifecycle stage and context.	Boundary of non-routine and experimentation activities well defined.	Autonomy of improvisation without detailed approvals within the defined task boundaries.
Switching		Swapping process methods across iterations, and lifecycle stages.	Activity and task-level accounting of resource allocation.	Flexibility in iteration lengths (sprints) and planning games.
Enrichment		Process-based learning and frequent on-the-job peer-level training.	Tracking of individual team member-level skill set and capabilities.	Highly capable and cohesive pair-programming teams.
Client Involvement and Relationship	Gronbaek *et al.* (1993); Leonard and Rayport (1997)	Follow empathic and participatory, user-centered design principles.	Bounded targets for Service Level Agreements (SLA) and traceability of performance.	Frequent involvement of end-users and collective decision-making.
Artifact Specification				
Requirements	Gomaa and Scott (1981); Davis *et al.* 1997); Morgan (2006, pp.110-11)	"Living requirements" with iterative prototyping.	Clear and sufficient documentation for reference.	Constant review of end-user requirements and tacit specifications.
Design		Model-driven and aspect-oriented specifications.	Scope boundaries of functional and non-functional requirements well defined.	Collective ownership of design (i.e., responsibility of all programmers) and constant refactoring possible without jeopardizing customer needs.
Project Governance				
Measurement and Performance Management	Fenton and Pfleeger (1998); Gopal *et al.* (2002, 2005)	Common platform and uniform and standardized procedures for metrics collection across process designs.	Statistical quality control and independence of SEPG and production teams ensured.	Flexibility of metrics design; simple, unobtrusive collection of metrics.
Controls	Kirsch (1997); Harris *et al.* (2009); Maruping *et al.* (2009)	Emergent Outcome Controls.	Objective, data-based comparison of outcomes and plans that are traceable to tasks.	Frequently evolving and continuous targets for project outcomes.
Volatility and Change Management	Barry *et al.* (2006); Vidgen and Wang (2009)	Change requests part of "living requirements" and negotiations bounded by the specifications and SLA targets.	Changes logged and recorded for independent analysis.	Simplified negotiations for changes enhance efficiency of empathic design and frequent end user-involvement.
Critical Incident Management	Keil (1995)	Joint escalations (client-vendor) to upper management.	Active involvement and verification by senior management.	Collective ownership of problems between end-users and programmers.

Artifact Specification

Documenting specifications that establish the common ground for different stakeholders in a software development project including end-users, programmers, architects, testers, and project managers needs to accommodate multiple viewpoints and is often resource intensive (Davis *et al.* 1997). Teams balanced the plan-based process requirement for reliable and comprehensive documentation with the agile process emphasis on working software by adopting mechanisms that facilitated "minimum critical specifications" (c.f., Morgan 2006, pp. 110-111). Some of the mechanisms that came up in our discussions included model-driven and aspect-oriented specifications, and iterative prototyping. Utilizing these specification tools and mechanisms in lieu of comprehensive natural language documents, teams were able to adequately create a common project reference source and also facilitate responsive action through continuous refinement, refactoring, and functionality change without bureaucratic negotiations.

Project Governance

As mentioned earlier, the teams were able to leverage process maps to implement task-level accounting of expenditures and extensive traceability mechanisms required by the organizational metrics program prevalent at the research site. As reported in prior research (Gopal 2002; 2005), the metrics program formed the basis of a rigorous performance measurement regime that helped institute relevant project controls in the projects. With a disciplined and structured approach enforced by the metrics program, process maps helped establish flexibility at the project level, and facilitated teams pursuing ambidextrous process designs to institute metrics that were relevant and meaningful to the project context. Project tracking and reporting were tailored for the partitioning mechanisms used by the teams and were used by the SEPG to derive organization-wide benchmarks for comparison and statistical quality control mechanisms. Thus, a rebalancing act between plan-based and agile components of the project-level process design did not disrupt functioning of project governance mechanisms. A project manager's comment illustrates the above result:

> "Metrics collection is not a separate effort for us. It is part of the process. The only question was whether metrics can live our written vs. verbal lifestyle as we switch from waterfall to sprints [SCRUM iteration]. We know [that] we don't want it to be your word versus my word in our client meetings, planning meetings, or in our sprint standup meetings. Reportable metrics that map to the client needs is thus very important. We only collect what we think are useful for our meetings, and we report what we use."

Similar to the findings reported by Harris *et al.* (2009), we noticed that the teams pursuing ambidextrous process designs used data from the metrics program to institute effective emergent outcome controls. The portfolio of emergent outcome controls included scope boundaries and iterative feedback mechanisms such as temporal partitioning of problem frames and minimum critical specifications that kept spiraling towards final deliverable working software. The rigorous measurement and performance

management mechanisms along with a portfolio of emergent outcome controls provided the necessary infrastructure for the teams and external stakeholders for data-driven decision-making on changes and critical incidents, and helped forge an environment of collective ownership. Changes were logged and monitored for independent analysis by both the internal and external stakeholders, and when necessary, they were used for bounding the limits of service level agreements. Committees involving members at the same functional level from the development and client teams handled critical incidents, facilitating joint-escalation schemes, which reduced reporting bias and coordination impediments (Keil 1995). The following comments from a team leader provides a narrative illustration:

> "When we started using XP with SCRUM for CMMI Level-5 KPAs, the real tension points were the reports for [our] account and delivery managers. Account managers and delivery managers want reports because they make decisions when things go bad — when we miss a delivery or when [customer satisfaction] rating is low. But, they don't have time for daily standup meeting reports and verbal satisfaction scores are not good enough. [Client] users have the same problem too. They have their managers to report to. So, we came up with a simple checklist that we jointly fill in our planning and standup meetings. The checklist is collated by quality assurance for account and delivery managers. When there are problems we go to managers to negotiate as one team…"

In summary, teams that pursued ambidextrous process design at the research site were able to achieve the discipline and control requirements of plan-based processes and the flexibility and improvisation focus of agile processes using several balancing tactics. Teams overcame typical impediments to ambidextrous designs by leveraging peer-learning, relational capital, emerging specification techniques and project governance modes that forged collective ownership and data-driven decision making.

6. Discussion

To shed an alternative perspective on the ongoing debate between plan-based and agile software processes, we examined the notion of software process ambidexterity and assessed the value of ambidextrous software process designs. The results from our analysis establish the significant and positive causal linkage between software process ambidexterity and a variety of project performance measures. Based on the empirical results from detailed project-level data and our qualitative observations, we now proceed to discuss managerial implications, especially the different steps that are needed in a pathway towards achieving software process ambidexterity in firms.

6.1. Pathway to Software Process Ambidexterity

The nested cycles of the needs of SEPG and project personnel that influence the characteristics of realized software process design is a key source of tension between the different organizational groups involved in software production. While organizational-level SEPG personnel desire standardization and control for better predictability, context-specific improvisations and adaptations are desired by project-

level personnel to aid better customer-orientation and cope with environmental volatility (c.f., Austin and Devin 2009; Barry *et al.* 2006; Lee *et al.* 2006). Figure 3 depicts a generalized representation of the nested cycles of needs, which we uncovered from our analysis of project data and qualitative analysis. The nested cycles move across the process design continuum with an agile methods emphasis on the one end and plan-based emphasis on the other. The corresponding coordination and governance mechanism enforced by the SEPG is thinner (lesser resources spent on monitoring and control) near the agile methods end of the process design continuum and relatively thicker (more resources spent on monitoring and control) on the plan-based methods end of the process design continuum. The characteristics of the projects selected and executed by the firm (i.e., market-driven environmental factors) pull and push the nested cycles (and the associated tensions) between SEPG and project-level teams across the software process design continuum axis.

The first necessary step to turn the nested tensions depicted in Figure 3 into a virtuous cycle of ambidexterity in software firms is to create the appropriate structural and operational support mechanisms (c.f., Andriopoulos and Lewis 2008). Structural support mechanisms ensure adequate "separation of concerns" between organizational-level governance personnel and project management personnel, and provide the necessary autonomy for independent action. Operational support mechanisms provide the necessary loose-coupling coordination routines between these autonomous units, enabling a common support region for independent, yet interrelated action. For example, at our research site the SEPG personnel and project managers were on the same hierarchy, acted independently without bureaucratic micro-management of each other's functioning. The control and experimentation design space of the SEPG personnel and the project managers respectively were constrained by a forcing common platform of governance function instituted by the firm's cross-functional centralized productivity office, which directly came under the purview of senior management (CEO's office). Such structural and operational support mechanisms help firms create an environment conducive for software production using ambidextrous software process designs.

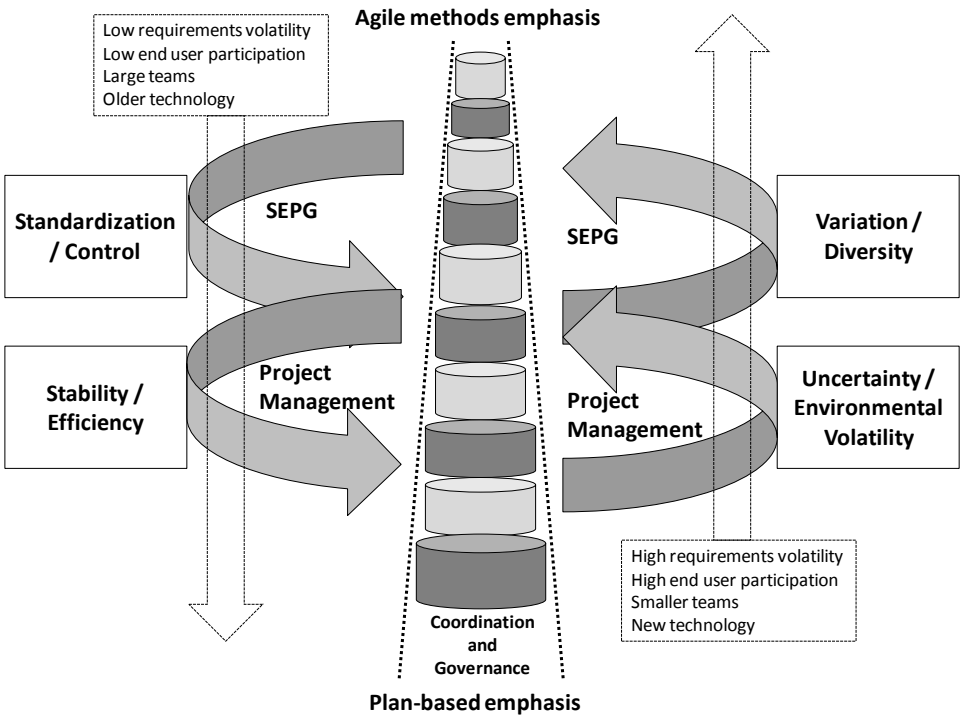

Figure 3. Nested Cycles of Software Process Ambidexterity

The second important step in the pathway to achieve organization-wide adoption of ambidextrous software process designs is the implementation of a rigorous organization-wide metrics program (e.g., Gopal 2002; 2005), which enables detailed observation of the micro-variations induced by ambidexterity such as the way project teams react to uncertainties and environmental volatility, and their corresponding performance outcomes. A comparative analysis of the project-level micro-variations across the process design continuum is facilitated by the organization-wide metrics program through a common measurement framework, and helps in the continuous refinement of software development meta-routines. For example, at our research site, irrespective of the process design choice chosen by a project team, detailed metrics (that were used in this study) were collected and stored in a central database for analysis. By deducing patterns from these micro-variations, best practices can be identified and institutionalized across the organization (c.f., Ramasubbu *et al.* 2008).

The third and final step in the pathway to achieve sustainable process ambidexterity in software firms is to recognize that not all projects benefit equally by applying ambidextrous process designs for software production. Our results indicate that the natural propensity of project teams to adopt (or avoid) ambidextrous software process designs may sometimes run counter to the best practices identified from studying the project-level metrics data. While a voluntary choice of process designs by project teams enables experimentation and learning-by-doing, it might also be necessary to curtail known sub-optimal choices through a combination of educational programs and portfolio of process controls. Advances in

implementing effective process traceability mechanisms and integrating knowledge management practices with the regular activities of a software development lifecycle (e.g., Ramesh 2002; Rus and Lindvall 2002) pave way for mechanisms that could curtail sub-optimal choices in software organizations that deploy ambidextrous process designs.

6.2. Limitations and Future Research

There are several limitations of this study that future research could address. First, since we observed only custom (bespoke) software development projects, we should be cautious in generalizing our results across all software development projects (maintenance, reengineering, product development, etc). The research methodology and the potential outcomes empirical analysis utilized in this study can be replicated in other software development settings and future research could embark on such replication and comparative analysis. Second, we did not observe the long-term impacts of ambidextrous process design and only studied the impact on immediate project performance outcomes. Future research could examine the long-term impacts of software process ambidexterity on learning curves and capability development of project teams. Third, our empirical context was limited to the examination of process evolution from a standardized plan-based starting point to an ambidextrous process design. Other variations of the process design evolution (for example movement from agile processes to ambidextrous process) need to be investigated and results compared with our findings. Finally, the distinct coexistence of several process designs in a software production ecosystem and the way these diverse process designs can be mapped to each other, controlled, and governed warrants further examination. We believe that these are fruitful lines of enquiry for future research on software process ambidexterity.

7. Conclusion

Reconciling the opposing approaches of plan-based and agile software process designs, we advanced the notion of software process ambidexterity and examined the antecedents and consequences of process designs that promote flexibility and improvisation without compromising discipline and control. The large-sample empirical results reported in the study establish the value of ambidextrous process designs. Our in-depth qualitative analysis of the balancing tactics employed by projects pursuing ambidextrous process designs shows that the common impediments to tradeoff shifting mechanisms can be overcome when ambidextrous rebalancing of plan-based and agile process components is hinged on client relationships, project governance, and efficient artifact specifications. We believe that this study lays a good foundation to "move beyond the entrenched disagreements about planning versus agility" (Austin and Devin 2009) and establishes a rigorous case for usefully combining the disparate control and flexibility-focused components of software development methodologies to create a new generation of process innovations in the hypercompetitive software industry.

References

Adler, P.S., Goldoftas, B., and Levine, D. I. 1999. "Flexibility versus Efficiency: A case study of model changeovers in the Toyota production system," *Organization Science* (10:1), pp. 43-68.

Aaen, I. 2003. "Software Process Improvement: Blueprint versus Recipes," IEEE Software (20:5), pp. 86-93.

Andriopoulos, C. and Lewis, M. W. 2008. "Exploitation-Exploration Tensions and Organizational Ambidexterity: Managing Paradoxes of Innovation," Organization Science (20:4), pp. 696-717.

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

Barry, E., Kemerer, C. F., and Slaughter, S. A. "Environmental Volatity, Development Decisions, and Software Volatility: A Longitudinal Analysis," *Management Science* (52:3), pp. 448-464.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunnigham, W., Fowler, M., Grenning, J, *et al.* 2001. "Manifesto for Agile Software Development," Accessed October 16, 2010, http://agilemanifesto.org/.

Begel, A., and Nagappan, N. 2007. "Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study," in *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, Madrid, Spain, September 20-21.

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

Boehm, B. and Turner, R. 2003a. "Using Risks to Balance Agile and Plan-Driven Methods," *IEEE Software* (36:6), pp. 57-66.

Boehm, B., and Turner, R. 2003b. *Balancing Agility and Discipline: A Guide for the Perplexed*, Boston, MA: Addison-Wesley Professional.

Boh, W. F., Slaughter, S. A., and Espinosa J. A. 2007. "Learning from Experience in Software Development: A Multilevel Analysis," *Management Science* (53:8), pp. 1315-1331.

Brown, S. L., Eisenhardt, K. M. 1997. "The Art of Continuous Change: Linking Complexity Theory and Time-paced Evolution in Relentlessly Shifting Organizations," *Administrative Science Quarterly* (42:1), pp. 1-34.

Cataldo, M., and Nambiar, S. 2009. "On the Relationship Between Process Maturity and Geographic Distribution: An Empirical Analysis of Their Impact on software Quality," *Proceedings of the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, Amsterdam, Netherlands, August 24-28.

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

Cox, B. J. 1990. "Planning the Software Industrial Revolution," *IEEE Software* (7:6), pp. 25-33.

Cusumano, M. A. 1994. "The Limits of "Lean"," *Sloan Management Review* (35:4), pp. 27-32.

Davis, A., Jordan, K., and Nakajima, T. 1997. "Elements Underlying the Specification of Requirements," *Annals of Software Engineering* (3:1997), pp. 63-100.

Dehejia, R. H., and Wahba, S. 2002. "Propensity score-matching methods for nonexperimental causal studies," *Review of Economics and Statistics* (84:1), pp.151-161.

Dijkstra, E. 1972, "The Humble Programmer," ACM Turing Lecture, Dikjstra Archives: http://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html, Accessed October 16, 2010.

Duncan, R. 1976. "The Ambidextrous Organization: Designing Dual Structures for Innovation," in Killman, R. H., Pondy, L. R., and Sleven, D. (eds.), *The Management of Organization* (1: 167-188), New York: North Holland.

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

Fenton, N. E., and Pfleeger, S. L. 1998. *Software Metrics: A Rigorous and Practical Approach*. Boston, MA: PWS Publishing.

Feldman, M. S., and Pentland, B. T. 2003. "Reconceptualizing Organizational Routines as a Source of Flexibility and Change," *Administrative Science Quarterly* (48:1), pp. 94-118.

Fichman, R. F., and Kemerer, C. F. 1997. "The Assimilation of Software Process Innovations: An Organizational Learning Perspective," *Management Science* (43:10), pp.1345-1363.

Galliers, R. D., and Swan, J. A. 1997. "Against Structured Approaches: Information Requirements Analysis as Socially Mediated Process," *Proceedings of the 13th Hawaii International Conference on System Science*, Wailea, HI, January 7-10.

Gibson, C. B., and Birkinshaw, J. 2004. "The Antecedents, Consequences and Mediating Role of Organizational Ambidexterity," *Academy of Management Journal* (47:2), pp. 209-226.

Gomaa, H., and Scott, D. 1981. "Prototyping as a Tool in the Specification of User Requirements," *Proceedings of the 5th International Conference on Software Engineering*, San Diego, CA.

Gopal, A., Krishnan, M .S., Mukhopadhyay, T., and Goldenson, D. R. 2002. "Measurement Programs in Software Development: Determinants of Success," *IEEE Transactions on Software Engineering* (28:9), pp. 863-875.

Gopal, A., Mukhopadhyay, T., and Krishnan, M .S. 2005. "The Impact of Institutional Forces on Software Metrics Program," *IEEE Transactions on Software Engineering* (31:8), pp. 679-694.

Gronbaek, K., Grudin, J., Bodker, S., Bannon, L. 1993. "Achieving Cooperative System Design: Shifting from a Product to a Process Focus," *Participatory Design: Principles and Practices*, eds. Schuler, D., and Namioka, A., Hillsdale, NJ: L. Erlbaum Associates Inc.

Hall, A.D. 1962. *A Methodology for Systems Engineering*, Reinhold: Van Nostrand.

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

Harter, D., Krishnan, M. S., and Slaughter, S. A. 2000. "Effects of Process Maturity on Quality, Cycletime, and Effort in Software Product Development," *Management Science* (46:4), pp. 451-466.

Harter, D., and Slaughter, S. A. 2003. "Quality Improvement and Infrastructure Activity Costs in Software Development: A Longitudinal Analysis," *Management Science* (49:6), pp. 784-800.

Hendricks, K. B., and Singhal, V. R. 1997. "Does Implementation of an Effective TQM Program Actually Improve Operating Performance? Empirical Evidence from Firms That Have Won Quality Awards," *Management Science* (43:9), pp. 1258-1274.

Highsmith, J. 2002. *Agile Software Development Ecosystems*. Boston, MA: Addison-Wesley Longman Publishing Co Inc.

Humphrey, W. S. 1989. *Managing the Software Process*. Reading, MA: Addison-Wesley

IEEE SPA. 2010. IEEE Computer Society Software Process Achievement Award, http://www.computer.org/portal/web/awards/processachievement, Accessed 17 October, 2010.

ISO9241-210. 2010. "Ergonomics of Human-System Interaction: Human-centered Design for Interactive Systems," International Organization for Standardization.

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

Kirsch, L. S. 1997. "Portfolios of Control Modes and IS Project Management," *Information Systems Research (8:3)*, pp. 215-239.

Krishnan, M. S., and Kellner, M. I. 1999. "Measuring Process Consistency: Implications for Reducing Software Defects," *IEEE Transactions on Software Engineering* (25:6), pp. 800-815.

Krishnan, M. S., Kriebel., C. H., Kekre, S., and Mukhopadhyay, T. 2000. "An Empirical Analysis of Productivity and Quality in Software Products," *Management Science* (46:6), pp. 745-759.

Lee, C-H., Venkatraman, N., Tanriverdi, H., Iyer, B. 2010. "Complementarity-based Hypercompetition in the Software Industry: Theory and Empirical Test, 1990-2002," *Strategic Management Journal* (31:13), pp.1431-1456.

Lee, G., Delone, W., Espinosa, A. J. 2006. "Ambidextrous coping strategies in globally distributed software development projects," *Communications of the ACM* (49:10), pp. 35-40.

Lee, G., DeLone, W. H., and Espinosa, A. J. 2007. "Ambidexterity and Global IS Project Success: A Theoretical Model," *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, Big Island, Hawaii, U.S.A., January 03-06.

Lee, G., Espinosa, A. J., and DeLone, W. H. 2009. "The Effect of Process Ambidexterity on the Success of Distributed Information Systems Development," *Proceedings of the Academy of Management*, Chicago, IL., U.S.A, August 07-11.

Leonard, D., and Rayport, J. F. 1997. "Spark Innovation Through Empathic Design," *Harvard Business Review* (75:6), pp. 102-113.

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

Mithas, S., and Krishnan, M.S. 2009. "From Association to Causation Via a Potential Outcomes Approach," *Information Systems Research* (20:2), pp. 295-313.

Morgan, G. *Images of Organization.*Thousand Oaks, CA: SAGE Publications Inc.

Nerur, S., Mahapatra, R., Mangalaraj, G. 2005. "Challenges of Migrating to Agile Methodologies," *Communications of the ACM* (48:5), pp. 73-78.

O'Reilly, C. A., and Tushman, M. L. 2004. "The Ambidextrous Organization," *Harvard Business Review* (82:4), pp. 74-81.

Parnas, D. 2006. "Agile Methods and GSD: The Wrong Solution to An Old but Real Problem," in Agerfalk, P. J., and Fitzgerald, B (eds.), Flexible and Distributed Software Processes: Old Petunias in New Bowls, *Communications of the ACM* (49:10), pp. 27-34.

Paulk, M. C. 2001. "Extreme Programming from a CMM Perspective," *IEEE Software* (18:6), pp. 19-26.

Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. 1993. "Capability Maturity Model, version 1.1," *IEEE Software* (10:4), pp. 18-27.

Puranam, P., Singh, H., Zollo, M. "Organizing for Innovation: Managing the Coordination-Autonomy Dilemma in Technology Acquisitions," *Academy of Management Journal* (49:2), pp. 263-280.

Raisch, S., and Birkinshaw, J. 2008. "Organizational Ambidexterity: Antecedents, Outcomes, and Moderators," *Journal of Management* (34:3), pp. 375-409.

Raisch, S., Birkinshaw, J., Probst, G., Tushman, M. L. 2009. "Organizational Ambidexterity: Balancing Exploitation and Exploration for Sustained Performance," *Organization Science* (20:4), pp. 685-695.

Ramesh, B. 2002. "Process Knowledge Management with Traceability," *IEEE Software* (19:3), pp. 50-52.

Ramasubbu, N., Mithas, S., Krishnan, M. S., and Kemerer, C. F. 2008. "Work Dispersion, Process-Based Learning, and Offshore Software Development Performance," *MIS Quarterly* (32:2), pp. 437-458.

Rosenbaum, P.R., and Rubin, D. B. 1983. "The Central Role of the Propensity Score in Observational Studies for Causal Effects," *Biometrika* (70:1), pp. 41-55.

Rosenbaum, P.R. 1984. "From Association to Causation in Observational Studies: The Role of Strongly Ignorable Treatment Assignment," *Journal of the American Statistical Association* (79:385), pp. 41-48.

Rubin, D. B. 2005. "Causal Inference Using Potential Outcomes: Design, Modeling, Decisions," *Journal of American Statistical Association* (100:469), pp. 322-331.

Rus, I., and Lindvall, M. 2002. "Knowledge Management in Software Engineering," *IEEE Software* (19:3), pp. 26-38.

SEI-PARS. 2010. "Software Engineering Institute – Published Appraisal Results," http://sas.sei.cmu.edu/pars/pars.aspx and http://www.sei.cmu.edu/cmmi/casestudies/profiles/pdfs/upload/2010SepCMMI.pdf , Accessed 16, October 2010.

Schmalensee, R. 2000. "Antitrust Issues in Schumpeterian Industries," *American Economic Review* (90:2), pp. 192-196.

Slaughter, S. A., Levine, L., Balasubramaniam, R., Pries-Heje, J., and Baskerville, R. 2006. "Aligning Software Processes with Strategy," *MIS Quarterly* (30:4), pp. 891-918.

Succi, G. 2010. "Agile Methods: Between Categorical Imperatives and Lean Production," in Agerfalk, P. J., and Fitzgerald, B (eds.), Flexible and Distributed Software Processes: Old Petunias in New Bowls, *Communications of the ACM* (49:10), pp. 27-34.

Turk, D., France, R., and Rumpe, B. 2002. "Limitations of Agile Processes", *Proceedings of the Third International Confernce on eXtreme Programming and Agile Processes in Software Engineering*, Alghero, Italy, May 26-29.

Vinekar, V., Slinkman, C. W., and Nerur, S. 2006. "Can Agile and Traditional Systems Development Approaches Coexist? An Ambidextrous View," *Information Systems Management* (23:3), pp. 31-42.