Communications of the Association for Information Systems

CAIS

Overview and Guidance on Agile Development in Large Organizations

Jordan B. Barlow
Operations and Decision Technologies Department, Kelley School of Business, Indiana University
jordy.barlow@gmail.com

Justin Scott Giboney
Department of Management Information Systems, Eller College of Management, University of Arizona

Mark Jeffrey Keith
Department of Computer Information and Decision Management, West Texas A&M University

David W. Wilson
Department of Entrepreneurship and Information Systems, Washington State University

Ryan M. Schuetzler
Department of Management Information Systems, Eller College of Management, University of Arizona

Paul Benjamin Lowry
Department of Information Systems, City University of Hong Kong

Anthony Vance
Information Systems Department, Marriott School of Management, Brigham Young University

Abstract:

A continual debate surrounds the effectiveness of agile software development practices. Some organizations adopt agile practices to become more competitive, improve processes, and reduce costs. Other organizations are skeptical about whether agile development is beneficial. Large organizations face an additional challenge in integrating agile practices with existing standards and business processes. To examine the effects of agile development practices in large organizations, we review and integrate scientific literature and theory on agile software development. We further organize our theory and observations into a framework with guidelines for large organizations considering agile methodologies. Based on this framework, we present recommendations that suggest ways large organizations with established processes can successfully implement agile practices. Our analysis of the literature and theory provides new insight for researchers of agile software development and assists practitioners in determining how to adopt agile development in their organizations.

Keywords: agility, agile development, software development, life cycle, large organizations, waterfall method, extreme programming, Scrum, informal communication, interdependencies, coordination

Volume 29, Article 2, pp. 25-44, July 2011

Electronic copy available at: https://ssrn.com/abstract=1909431

I. INTRODUCTION

A continual debate surrounds agile software development practices. Agile software development refers to a development methodology that uses iterative development, frequent consultation with the customer, small and frequent releases, and rigorously tested code [Cao et al., 2009]. Some organizations adopt agile practices to become more competitive, improve processes, and reduce costs. Other organizations are skeptical about the benefits of agile development. Large organizations face an additional challenge in the integration of agile practices with existing standards and business processes.

Software developers created agile development largely to address the weaknesses of plan-based methods of software development, such as the influential waterfall method [Royce, 1970]. The primary weakness of plan-based methods is a lack of responsiveness to change. Due to the sequential nature of plan-based development, developers using this methodology establish requirements and plan projects early in the process, resulting in reduced flexibility in subsequent phases of development. However, project requirements often change significantly between initiation and completion. Agile development enables organizations to adapt to these dynamic conditions, facilitating more flexible development [Cockburn and Highsmith, 2001].

However, critics doubt whether the benefits of agile development outweigh the costs [Ambler, 2008; Rising and Janoff, 2000; Selic, 2009]. Most of these critics point specifically to a lack of focus on planning and implementation, with too much focus on coding [Ambler, 2008]; a lack of needed documentation that often results from decreased formal communication [Selic, 2009]; and implementation failures in larger, more complex projects [Rising and Janoff, 2000].

To examine this debate from the viewpoint of large organizations, we review and integrate literature on agile software development. We present a brief description of agile software development, its general strengths and weaknesses, and the organizational changes required to implement agile methods. Many research articles explain the benefits and weaknesses of agile development in small or simulated environments. However, little research examines the impact of agile development on large organizations. In addition, few researchers use a theory-based approach to examine the reasons for success or failure of agile techniques in large organizations. In this article, we present a theory-based framework and recommendations that suggest ways large organizations with established processes can successfully implement agile practices.

Our summary of the literature, as well as our theory-based framework and associated recommendations, provide insight for researchers of agile software development and assist practitioners in determining how to adopt agile development in their organizations. We elaborate upon situations in which either agile or traditional methods might be better suited. Further, we recommend the implementation of an agile-traditional hybrid method. Hybrid methods allow traditional software development teams to implement some practices of agile development to build on the strengths of their existing methods.

This article is organized as follows: Section II reviews the background of agile software development. Section III discusses the theoretical basis for outcomes of agile methodologies. Section IV covers the theory-based implications of agile development methods in large organizations. Section V presents our framework for choosing an agile software development methodology depending on project conditions. Section VI addresses the suitability of and implementation strategy for agile development methods specific to large organizations. Finally, Section VII suggests ideas for future research, and Section VIII provides conclusions.

II. BACKGROUND ON AGILE SOFTWARE DEVELOPMENT

Development life cycles define the way a group develops software. Life cycles can determine the project leader, the number of participants, the frequency and formality of team communication, the primary objective, and other facets of system development. Because a team's particular life cycle determines so much of what happens during a project, developers often seek to improve life cycle methods or to implement more effective methods.

One major problem developers often face is the need to adapt to changes [Austin and Devin, 2009]. Requirements often change after a project begins, and customers and sponsors often change their expectations for the final product. Traditional development methods usually include provisions for responding to changing requirements, but

Electronic copy available at: https://ssrn.com/abstract=1909431

Communications of the Association for Information Systems

these provisions take time and can be costly. As a result, many developers adapt life cycles that allow increased agility.

Researchers define *agility* in various ways, and views on the concept often conflict [Sarker et al., 2009]. Information Systems literature defines agility as "the continual readiness of an [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), through its collective components and relationships with its environment" [Conboy, 2009, p. 377]. This idea of creating more agile methods became widely popular in the 1990s [Austin and Devin, 2009].

In 2001, a group of software developers met to establish the primary principles of agile development, which we summarize in Table 1. In each case, agile developers consider the first item listed as more valuable than the second. For example, proponents of agile methods value documentation, but not as highly as they value working software: "We embrace documentation, but not hundreds of pages of never-maintained and rarely-used tomes" [Beck et al., 2001].

Table 1: Agile Principles [Beck et al., 2001]

Agile Principle	Description
Customer collaboration over contract negotiation	Reduce formalities to start and finish faster, with a strong focus on the customer throughout the development process
Individuals and interactions over processes and tools	Enhance communication within teams and barrier removal
Working software over comprehensive documentation	Developers spend more time coding and testing than they do writing extensive documentation
Responding to change over following a plan	Give teams the freedom to make changes and adjust to project needs

While some developers seek to integrate agile principles in existing methodologies, others create formalized agile life cycles, the most popular being Scrum and eXtreme Programming (XP) [Conboy, 2009]. Each agile methodology employs unique practices while maintaining focus on the core principles shown in Table 1. For example, Scrum focuses on project management aspects of development, using short, frequent team meetings to assess progress. Conversely, XP focuses on the development process itself by prescribing specific techniques such as pair programming.

Agile development methodologies are based on "intensely iterative processes" [Austin and Devin, 2009, p. 463], meaning that teams analyze, design, and code rigorously in short intervals; meet with the customer to evaluate their progress and get feedback; and start the cycle again. This method contrasts sharply with traditional methods, where a long design phase precedes a long coding phase, which in turn precedes a long testing phase.

Boehm and Turner [2005] state that a truly agile process must also be self-organizing and emergent. Self-organizing means that teams make decisions through informal communication and frequent, short meetings rather than relying on one owner to guide the project. Emergent in agile development indicates that requirements emerge during the course of the project, with almost no time spent on design before coding starts.

The primary difference between agile and traditional development paradigms is that agile paradigms focus on adapting well to changing project requirements [Austin and Devin, 2009]. Advocates of plan-based methods argue that time spent on design is key to developing a good system that meets specifications, and that reduced flexibility is a necessary trade-off. In contrast, proponents of agile methods argue that a system developed using an iterative method can still meet all requirements and proper design standards, while frequent iterations allow for repeated refinement and a final product that is closer to customers' actual desires.

Strengths and Weaknesses

A large body of literature highlights the strengths of agile methods. First, as mentioned, agile methods allow flexibility and adaptability [Austin and Devin, 2009]. Because the design phase is not formal and programmers work in short intervals on smaller milestones, significant requirements of the project can change as development proceeds without much loss of productivity.

Agile methods also promote a focus on customer needs. Because of the adaptability of agile methods, customers' requested changes in plans or requirements can often be included in the next iteration of development. Practitioners

Electronic copy available at: https://ssrn.com/abstract=1909431

Communications of the Association for Information Systems

note that agile methods help them stay focused on customer needs and changing desires throughout the project [Cockburn and Highsmith, 2001].

Agile methods can also often lead to faster development, depending on project size and actualization of project risks. Without a detailed design phase or documentation throughout the project, the team can focus on the act of development—coding and testing the product. In a small- or medium-sized project where teams develop software with few iterations, agile development leads to a shorter development cycle than do plan-based methods.

Although agile methods are popular and successfully applied in small- and medium-sized projects, critics are quick to point out weaknesses of the methods. These weaknesses are often more influential in large or complex projects, or in large, structured firms. For example, AG Communication Systems, a company with development teams ranging in size from two to several hundred members, found that while small teams performed well with Scrum, agile methods were not as effective for large teams [Rising and Janoff, 2000].

With agile methods, development begins before the requirements are well defined. In complex and/or large projects, this approach is potentially crippling, because important features might be forgotten or misunderstood, requiring additional work later in the project. Likewise, time and resources can be hard to estimate without a detailed plan. The short iterations of agile development are intended to provide adaptability and customer focus. However, without a detailed planning phase, a fairly accurate estimate of resources and time requirements is virtually impossible, since the customer is enabled to add or remove features during the development process. For small- or medium-sized projects, this uncertainty is an acceptable risk. For large or complex projects, the magnitude of uncertainty is greater, and constitutes a prohibitive risk for most organizations.

In large projects, agile methods do not promote formal lines of communication. One study pointed to the benefits of agile development in facilitating good communication among project team members, but also noted that, when using agile methods "in larger development situations involving multiple external stakeholders, a mismatch of adequate communication mechanisms can sometimes even hinder the communication" [Pikkarainen et al., 2008, p. 303].

In addition to decreased formal communication, scant documentation can be particularly detrimental to large or complex projects. Software products require periodic maintenance, and documentation facilitates maintenance. Many smaller projects, on the other hand, require less formal updating than large projects, and as such, are not as dependent on careful, detailed documentation.

We summarize the strengths and weaknesses of agile development in Table 2. Clearly, both agile and traditional approaches have strengths and weaknesses, and a single approach is not suited for every development project. Agile methods' strengths generally benefit smaller, less complex projects, and their weaknesses surface most evidently in large, complex projects. In these large, complex projects, the weaknesses of traditional methods help to mitigate risks associated with uncertainty and lack of structure. This article helps inform the decision, given a project's characteristics, of whether agile or traditional methods, or a combination of the two, would be best. Of course, organizations cannot eliminate all the risks associated with project uncertainty. The goal is to mitigate risk where possible, and to choose a method or methods that best accomplish that goal.

Table 2: Strengths and Weaknesses of Agile Development

Strengths	Weaknesses
Focus on customer needs	Does not promote formal communication
Adaptable to changing requirements	Time and resources might be unknown initially
Fast development time	Requirements not well defined
	Lack of documentation

Research on Agile Methodologies

A number of academics and practitioners in the information systems community have discussed the increasing popularity of agile development methods and the implications of that growth [e.g., Austin and Devin, 2009; Conboy, 2009; Sarker, 2009]. These articles seek to explore and explain the strengths and weaknesses of agile development listed above. Many articles [e.g., Mangalaraj et al., 2009; Vidgen and Wang, 2009] are concerned with purely agile methodologies such as XP and Scrum; other articles [e.g., Fitzgerald et al., 2006; Karlsson and Agerfalk, 2009; Port and Bui, 2009] discuss hybrid methodologies, a combination of agile and plan-driven methods discussed later in the paper. Appendix A contains a summary of literature concerning agile methodologies, along with the research methodologies used. This summary provides guidance for future research concerning agile and hybrid methodologies.

Electronic copy available at: https://ssrn.com/abstract=1909431

A review of literature on agile methodologies in large organizations reveals that most research is narrative in detail and consists mainly of case studies [Berger and Beynon-Davies, 2009; Boehm and Turner, 2005; Fitzgerald et al., 2006]. While many case studies give helpful ideas and recommendations, these recommendations are not widely generalizable because they lack theory development to fully explain their results. A small number of research articles examine theoretical aspects of agile development [e.g., Pikkarainen et al., 2008; Sarker et al., 2009], but these articles do not address our concern that agile methods have less success in large organizations or complex projects.

One exception is research by Cao et al. [2009], who use adaptive structuration theory to examine successful implementations of agile techniques by adapting development methodologies. Their study uses theory extensively to examinine agile techniques in large projects and teams. However, our study takes a different approach. While Cao et al. explain ways to adapt methodologies to specific circumstances, we aim to explain theoretically the reasons why certain types of environments are more suitable for agile techniques than others. Additionally, by examining the theory connected to success or failure in implementation of agile methods—particularly in large organizations or complex projects—we are able to create a guiding framework to aid developers in deciding between agile, plan-based, and hybrid software development methodologies.

III. THEORETICAL DEVELOPMENT

The need and motivation for agile techniques and their success or failure can be better understood by reviewing relevant organizational theory on interdependence and coordination [Thompson, 1967]. In his theory on organization structure, Thompson [1967] outlines three types of interdependencies and three types of coordination used to manage those interdependencies. The types of interdependencies present in an organization and the costs of interdependency coordination can determine, in part, the appropriate software development methodology.

Pooled interdependencies are those that arise from an individual being loosely grouped with other individuals and sharing common risks. For example, insurance customers share a pooled interdependency with all other customers because they each depend on a critical mass of contributors who can offset the risk of an accident. Similarly, every member of a software development team shares a pooled interdependency with each other member. In order for the project to be completed, every member must do his/her own part—though most members do not directly depend on the inputs or outputs from every other team member. Pooled interdependencies are the least costly to coordinate. Standardization is the coordination technique used to manage pooled interdependencies. For example, to ensure that every project team member does his or her part, organizations or teams create standards for the number of work hours required per week and the quality criteria for each team member's output.

Sequential interdependencies result from the serial nature of workflow. In software development, some form of analysis must take place before design, design before development, development before implementation, etc. In other words, if team member A's output is required as B's input, then B has a sequential dependency on A. The cost of coordinating sequential interdependencies is higher than for pooled interdependencies. Coordination of sequential interdependencies requires planning that takes place repeatedly and uniquely for each project, as opposed to standardization, which organizations establish once for several projects.

Last, reciprocal interdependencies exist when two or more parties depend on each other in both directions. Further, the nature of the dependency might be unclear initially. For example, consider a sequential dependency in which developer B requires the code produced by A. However, if B finds bugs or mistakes in A's code, then the code must be sent back to A. Therefore, a reciprocal dependency exists in which B depends on code from A, yet A also depends on B to test and approve A's output. Similarly, consider the scenario where developers A and B each produce a software module. B's module requires an input which is an output of A's module. In this case, B depends on A to produce an appropriate output and A depends on B to specify his/her required input.

Reciprocal interdependencies relate to high risk and uncertainty in project processes. For example, uncertainty in the software development process can be the result of unexpected changes to scope or shortcomings of project plans or competency of team members [Schmidt et al., 2001]. Reciprocal interdependencies typically require coordination in the form of mutual adjustment, which refers to the ad hoc, informal communication that takes place outside of formal project plans and often continually throughout a project [Thompson, 1967].

Plan-driven methodologies assume that project interdependencies are mostly sequential and can be managed through coordination in the form of planning and review. However, many interdependencies in a software project life cycle are actually reciprocal in nature. As a result, some of the time and cost spent on the creation of detailed plans is wasted and a certain degree of mutual adjustment is required. Comparatively, agile methodologies assume the opposite. They frame most or all project interdependencies as reciprocal and, therefore, adopt mutual adjustment to

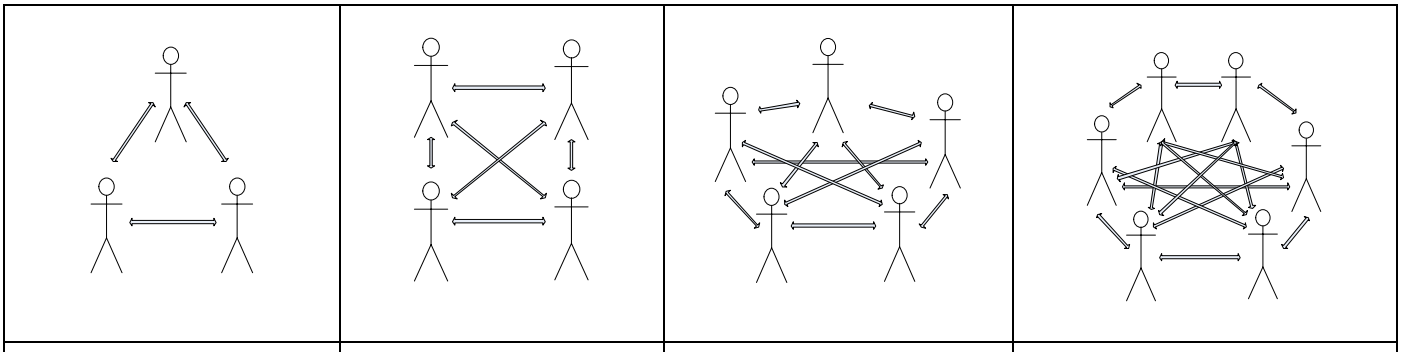
Electronic copy available at: https://ssrn.com/abstract=1909431

Communications of the Association for Information Systems

coordinate all project interdependencies. In other words, they de-emphasize formal, upfront planning and coordinate ad hoc as the needs arise. However, according to structural contingency theory [Donaldson, 2001; Thompson, 1967], the unnecessary cost of using mutual adjustment rather than planning for sequential interdependencies is a waste. Ideally, IT project teams would adopt methodologies using a hybrid coordination strategy that uses mutual adjustment only for reciprocal interdependencies and planning for sequential interdependencies. Accordingly, we base our recommendations on this theoretically ideal scenario.

In addition to using the appropriate coordination strategy for interdependencies that exist in the software development process, project teams must have the infrastructure in place to support their chosen method of coordination. For example, to make effective use of planning, project managers need to fully understand the nature of all sequential interdependencies, the scope of the individual tasks required, and the resources needed to complete those tasks. They must then be able to accurately estimate time and cost requirements, which requires a significant degree of knowledge and experience on the part of those creating the plans. Without this implicit knowledge, a system must exist to help new managers reuse the knowledge and experience developed from prior projects.

Conversely, to make effective use of mutual adjustment through informal knowledge and information sharing, a strong, well-connected social network must allow informal communication to take place. Thus, organizations should minimize the costs of communication through conditions such as collocation of team members or availability of appropriate communication media. This requirement explains, in part, why agile techniques are difficult with large groups—the number of additional social network connections that must be created to adequately incorporate a team member increases exponentially with each added actor to a network, as shown in Figure 1. In reality, most actors will not need to collaborate with every other actor. However, without well-defined and predictable roles, scope, and communication channels, every actor might *potentially* need to collaborate with any other actor.



3 actors = 3 relationship	4 actors = 6 relationships	5 actors = 10 relationships	6 actors = 15 relationships

Figure 1. The Complexity of Additional Social Network Actors

These theoretical implications are not enough to prescribe most managerial actions. Project managers might assume that poor performance on software development projects is the result of a mismatch between their chosen methodology and their project's interdependencies. For example, in a plan-driven environment, poor performance might not be the result of too much reciprocal interdependency but, rather, the result of a poor infrastructure to support project planning. In this situation, the best decision would be to improve the planning infrastructure rather than switch to an agile methodology. Similarly, poor performance in an agile environment might be the result of a weak social network rather than the presence of sequential interdependencies. Thus, making the optimal decision concerning a software development methodology requires an understanding of both the nature of a process's interdependencies as well as the ability to support the type of coordination required by that interdependency.

Information processing theory [Galbraith, 1973; Tushman and Nadler, 1978] posits that a firm's performance is based on achieving a "fit" between its information processing needs and its information processing capabilities. The uncertainty and interdependencies associated with core business processes generate information processing needs. An organization's information processing capabilities stem from a variety of sources, including its information technologies, formal organization structure, and ability to support informal coordination.

If an organization's information processing needs are low, its information systems and formal structure should be designed to support coordination in the form of planning and standardization. If information processing needs are high, managers should design information systems and formal structure to support informal communication and collaboration and have plenty of available slack resources. A mismatch between an organization's information

Electronic copy available at: https://ssrn.com/abstract=1909431

processing needs and its associated support structure results in either poor information processing performance or wasted resources (see Table 3).

Table 3: Information Processing Perspective [Galbraith, 1973; Tushman and Nadler, 1978]

		Information Processing Needs	
		High	Low
Information Processing Capabilities	High	Optimal Performance	Wasted Resources
	Low	Low Performance	Optimal Performance

Another potential mistake is to assume that the nature of project interdependencies is deterministic and unchangeable. Interdependencies arise from uncertainty [Thompson, 1967]. Thus, changes in uncertainty and risk within an organization lead to changes in interdependencies. Using the example of reciprocal interdependency above, developer A is reciprocally interdependent with B due to uncertainty of whether the program module will perform as planned without user testing. The uncertainty increases if the developers are inexperienced, less knowledgeable, or unused to working together. This type of risk becomes a greater threat in large organizations where turnover is high.

Organizations can minimize risk by optimizing the formal organization structure, creating self-contained, modular tasks [Galbraith, 1973; Thompson, 1967], or selecting communication media appropriate to the task [Dennis et al., 2008]. For example, managers should formally group those roles and individuals who naturally have the greatest reciprocal interdependencies to reduce the cost of their coordination [Thompson, 1967]. If managers also successfully modularize the software development process into independent, self-contained tasks, fewer reciprocal interdependencies will arise between groups of developers. Finally, if managers properly group roles and modularize tasks, they can select the appropriate information technology to support reciprocal coordination within groups and sequential coordination between groups. Organizations that successfully execute these strategies might be able to create preferable alternatives for software development methodology and supporting infrastructures.

IV. AGILE DEVELOPMENT IN MATURE ORGANIZATIONS

Most articles showcasing the success of agile life cycles use real-world scenarios as evidence, but these scenerios are typically in small or low-risk projects, often in smaller organizations [e.g., Cockburn and Highsmith, 2001; Kussmaul et al., 2004; Pikkarainen et al., 2008; Rising and Janoff, 2000]. The small projects described in these articles enable teams to support an infrastructure that fits project interdependencies. For example, in one case studied by Pikkarainen et al. [2008], the project team consisted of only six members. The tasks of team members in creating a security management system were reciprocally interdependent, which required constant informal communication between team members. Due to small team size, agile strategies such as open office spaces and informal team meetings successfully supported project needs. Thus, agile development is best suited for small- or medium-sized collocated teams [Boehm and Turner, 2003; Lindvall et al., 2004].

In contrast, large or complex projects and/or mature organizations are more problematic scenarios for implementing agile methodologies. Because large, complex projects and organizations naturally have many interdependencies, projects require a large amount of coordination. As a result, less costly forms of coordination—standardization and planning—are easier to control in these situations. In other words, most people in large projects and organizations find it hard to implement mutual adjustment coordination on a large scale. Because agile methods depend on mutual adjustment, these methods are usually not preferred for large, complex projects or mature organizations.

A complication specific to mature organizations relates to IT governance frameworks such as ITIL, CMMI, or COBIT.[1] These frameworks ensure alignment of IT with business goals and provide structure to IT development and management processes. In a typical agile environment, structure established by a governance framework might hinder project progress [Boehm and Turner, 2005]. We address this issue later in the article by suggesting the use of hybrid methodologies.

Researchers and practitioners who study or implement agile methods in large, mature organizations document the associated difficulties [e.g., Berger and Beynon-Davies, 2009; Boehm and Turner, 2005; Lindvall et al., 2004; Pikkarainen et al., 2008; Pikkarainen and Passoja, 2005]. Proponents of agile methodologies make suggestions for scaling agile methodologies to larger teams and projects [e.g., Cohn, 2007], but in our review of academic and practitioner journals, we find no articles with complete success stories of agile life cycles in large organizations (see Appendix A).

[1] Acronym definitions: Information Technology Infrastructure Library (ITIL), Capability Maturity Model Integration (CMMI), and Control Objectives for Information and related Technology (COBIT).

Electronic copy available at: https://ssrn.com/abstract=1909431

Communications of the Association for Information Systems

Clearly, agile development in its strictest form is likely not a good solution for many development initiatives at large, mature organizations. As our reasoning suggests, teams and/or projects in the cases noted above were too large to fully support the informal nature of agile development methods. Though these projects might have entailed reciprocal dependencies, the infrastructure was not compatible for purely agile methodologies. In these cases where reciprocal dependencies are present but agile methodologies are not feasible, a hybrid methodology, which we will discuss in the next two sections, might be the best solution.

V. CHOOSING A METHODOLOGY

Based on our literature review and theoretical observations of difficulties with agile methods in large organizations, we present a framework for choosing a methodology appropriate to an organization's needs. Our framework is presented in Figure 2, which illustrates the recommended methodology, coordination strategy, and investment options to support coordination for project teams of varying size, interdependencies, and volatility. In Figure 2, we use the term volatility to refer to the instability associated with turnover in the project team. Turnover becomes more likely with drastic changes in the economy. When business is booming, companies hire more employees who need time to adapt to the organization's culture and develop appropriate skills. During recessions, many organizations will lay off higher-paid employees in favor of new college graduates, whom they can pay smaller salaries. Therefore, we differentiate our theoretical framework between high and low team volatility environments.

When most project interdependencies are sequential (Figure 2, Boxes A and D), regardless of team size or project volatility, the project manager should adopt a plan-driven methodology and invest in technologies that will support the project planning process. For example, knowledge management systems that can facilitate the easy storage and retrieval of past project plans, resources, key performance indicators, successes, failures, etc. would be useful. New project managers could step in, use documented information, and expect it to be a relatively accurate indicator of future results. Experience and training are less important in this environment because project managers can easily codify relevant project knowledge into the knowledge management system. Social network connectivity, which facilitates informal knowledge sharing, is also less important because most necessary lines of communication can be predicted, specified, and planned for upfront. As a result, team size is less relevant because coordination is taking place through project planning rather than during implementation through informal social networks.

Similarly, formal organizational structures do not need to be based on grouping reciprocal interdependencies and can follow other strategies, such as grouping high- and low-experienced individuals to promote knowledge creation and transfer [Nonaka, 1994] or grouping those with similar roles to promote economies of scope [Panzar and Willig, 1981]. The only differentiator when most project interdependencies are sequential is that organizations with low team volatility (Figure 2, Box D) might also consider investing in their project manager's competence and tacit knowledge. With fewer turnovers among project managers, organizations are more likely to realize the benefit of investing in tacit knowledge development.

When the nature of project interdependencies is reciprocal (Figure 2, Boxes B, C, E, and F), our theoretical recommendations vary depending on project team size and volatility. When project team size is small (Boxes C and F), we recommend adopting an agile methodology. Agile methods, with iterative cycles and frequent communication among team members and stakeholders, are well-suited to small teams with highly reciprocal interdependencies. However, the approach that project managers should take to support frequent communication varies depending on team volatility. If volatility is low (Box F), project managers should promote a strong social network for informal knowledge sharing, leaving fewer disruptive changes to the network structure. For example, by analyzing team members' personalities, demographics, and values [Klein et al., 2004], project managers can group individuals who are most likely to develop friendships and knowledge-sharing relationships. Similarly, project managers can improve social network connectivity by creating groups in which members share similar tasks or by providing training on appropriate types of collaborative technology [Keith et al., 2010].

If volatility is high (Figure 2, Box C), efforts to improve social networks might be unproductive due to turnover. Therefore, a project manager's best strategy for coordinating reciprocal interdependencies would be to group the roles that are most likely to be reciprocally interdependent [Thompson, 1967]. In that case, whoever joins the team to fill those roles will already be in close physical proximity and linked in the formal organizational structure.

Lastly, we recommend that organizations with mostly reciprocal interdependencies and large group sizes adopt a hybrid methodology (Figure 2, Boxes B and E). As will be discussed in the next section, a hybrid methodology requires managers to decompose project tasks into modules that are as independent as possible. Once the project is modularized, the manager can use plan-driven techniques for any project modules that have mostly sequential interdependencies, and agile techniques for the majority of modules that have reciprocal interdependencies. If such projects can be successfully modularized, project managers can use plan-driven techniques to coordinate the actions of sub-teams.

Electronic copy available at: https://ssrn.com/abstract=1909431

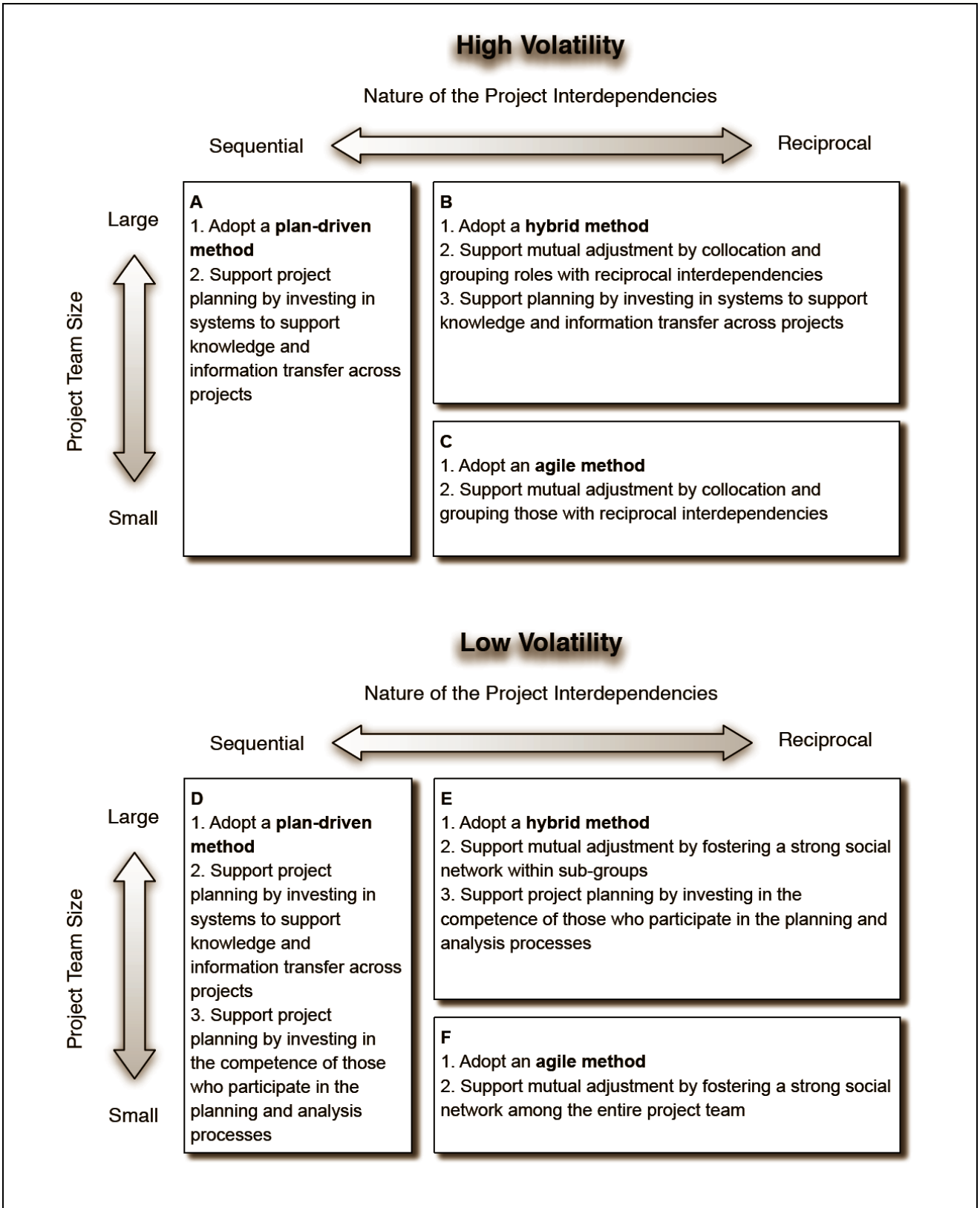


Figure 2. Methodology Selection Framework

For example, service-orientation and cloud computing paradigms help developers compose IT systems from loosely coupled, independent software modules or services [Bardhan et al., 2010]. If an organization can technically design a new system in such a manner, the project manager can naturally decompose the software development process around these modules. Project managers need not care whether each module is developed using a plan-driven or agile methodology—only that each software module accepts the proper inputs and produces the required outputs while meeting quality standards.

Electronic copy available at: https://ssrn.com/abstract=1909431

Communications of the Association for Information Systems

Although we recommend a hybrid approach for large organizations with reciprocal interdependencies, the appropriate infrastructure for supporting project coordination depends on the level of team volatility. When project turnover is low (Box E), organizations can promote social network connectivity with a greater level of certainty that their efforts will not be in vain. For the same reason, those organizations should also invest in their project manager's tacit knowledge development and general competence. Organizations should have both a well-connected advice network as well as a strong transactive memory among team members. Transactive memory refers to how well each project team member understands the strengths and expertise of other team members [Faraj and Lee, 2000]. In addition, project teams must have the ability and support structure to effectively share that knowledge and expertise across time and space since it becomes increasingly difficult to collocate the entire project team as it grows in size.

Conversely, when volatility is high (Box B), investment in social networks that are easily disrupted or in project managers who frequently change might not be worthwhile. Rather, the organization should focus on codifying as much knowledge as possible into a knowledge management system that will assist with project planning and sourcing. Those organizations should also group the roles which are most likely to be reciprocally interdependent. Importantly, the focus is on grouping roles rather than on individuals since actual employees who fill the roles frequently change.

We base our framework and implications for practice on relevant theory on organizations, structure, and technology [Galbraith, 1973; Thompson, 1967]. The key is to achieve an appropriate fit between interdependency, team size, and team volatility and the selected software development methodology and supporting infrastructure. Many of the salient factors identified in prior IT project research such as risk and coordination [Kraut and Streeter, 1995; Schmidt et al., 2001], are also based on these basic concepts.

While our suggestions stem from theory, they certainly might vary depending on the real costs of the three options. For instance, Fitzgerald et al. [2006] provide an example of a successful hybrid method adoption. The Intel Shannon teams in the case were relatively small, between four and six members. Our framework suggests that hybrid methods are most appropriate for larger teams, with smaller teams more suited for stricter forms of agile development. However, in a manner similar to larger teams and projects, Intel Shannon appears to weight its need for structure, formal communication, and compliance with governance frameworks more heavily than the benefits promised by agile methodologies. The organization adopted only portions of agile methodologies, preserving many of the risk-reducing strategies in its traditional methods. We intend our framework and recommendations to be a set of guidelines that can be adapted to individual situations according to specific needs of the organization.

VI. HYBRID METHODOLOGIES IN LARGE ORGANIZATIONS

Because a purely agile methodology is not suited for general use at a large, mature organization, we recommend, where appropriate, the implementation of a traditional/agile hybrid solution that will enable project teams to take advantage of the organization's maturity in software development while gaining advantages of agile development such as adaptability to changing requirements.

Support for the combination of agile and traditional development methods continues to grow in academic literature [Fitzgerald et al., 2006]. Hybrid solutions are often more successful than other methods in large organizations [Boehm and Turner, 2003; Cao et al., 2004; Cao et al., 2009]. By introducing traditional/agile hybrid methods, large organizations can take advantage of some benefits of agile development without abandoning the stability provided by traditional methods.

Portions of agile methodologies are successful alternatives to traditional development practices when used in situations described by our framework and theory. We base our recommendations primarily on the published results of actual experiences of various software development units. Several studies [Harris et al., 2009; Maruping et al., 2009a; Maruping et al., 2009b] show agile methods to be highly successful, particularly when requirements change during the life of the project. As mentioned earlier, some successful implementations of particular agile methods include projects at large organizations.

Examples of large companies that successfully use agile practices in hybrid life cycles for large projects include Nokia [Kahkonen, 2004] and Motorola [Bowers et al., 2002]. Kahkonen [2004] describes in detail three development styles used at Nokia in large projects. Nokia's philosophy is that large projects require large numbers of team members, but communication and coordination become difficult as teams get larger. The Kahkonen case highlights ways in which a large organization implemented only portions of XP to mitigate some of the negative effects of this increase in communication and coordination effort. This falls directly in line with our theoretical framework and

Electronic copy available at: https://ssrn.com/abstract=1909431

recommendations, in which we suggest a hybrid methodology for projects or teams that are large or complex, but for which some of the benefits of agile are still desired.

Large teams of developers in a highly structured department at Motorola implemented a hybrid life cycle using some practices from the XP methodology [Bowers et al., 2002].[2] This successful project contained fifteen different, reciprocally dependent pieces—indicating a need for mutual adjustment coordination in the form of agile practices. In the words of a Motorola employee, "we took XP, adopted some practices, dropped others, and supplemented others with practices from [traditional development methods]" (p. 100). To support the XP practices adopted, the organization collocated employees, rearranging cubicles to more easily support pair programming. These changes in infrastructure and methodology helped Motorola to achieve success on the project.

As noted previously, IT governance frameworks in place at large organizations might conflict with some agile methodologies. Such an organization should use only a few agile methods rather than an entire life cycle. For example, an organization of this type might benefit most from disregarding the agile principle of "no ownership" while implementing other agile methods, such as frequent iterations [Bowers et al., 2002]. This strategy allows existing structure to stay in place while other methods help to achieve agility in the project.

One area especially affected by change in development methodologies is release management. Release management is a major component of ITIL, and release management principles play a major role in other frameworks as well. Because release management usually focuses on introducing new or updated software, working with customers on planned releases and quality testing, agile methods alter the functions of the release management team. Increased time is necessary for communication with customers because agile methodologies require feedback from customers on a regular basis. In addition, quality testing is done on a more regular basis.

Several organizations already achieve process maturity in a governance framework while simultaneously using some agile methods in their development life cycles. For example, Intel Shannon [Fitzgerald et al., 2006] was assessed as a Level 2 on the Capability Maturity Model, which implies a measure of discipline and structure in the development process. The company experienced significant growth in its workforce (i.e., high volatility). Requirements analysis and actual development required constant collaboration between groups, implying that project interdependencies were reciprocal in nature. As our framework would indicate, management at Intel Shannon used a hybrid method, implementing various portions of both Scrum and XP methodologies. Managers eliminated other portions of these strategies to create a custom blend of agile methods that worked within their organizational structure—a strategy they termed *method engineering*.

Developing Hybrid Methodologies

As we have noted, development methodologies can include portions of agile methods in various ways. Because each organization and project is different, we next review and describe general techniques for developing hybrid methodologies. Organizations should first evaluate project size, volatility, and project interdependencies to determine whether a hybrid method is appropriate. If so, they could choose to use one or more of the following techniques to develop the actual methodology.

Adapted Base Methodology

Karlsson and Agerfalk [2009] suggest that organizations "want to achieve a method that fits the situation and at the same time aligns with the basic goals and values of the method" (p. 301). Teams choose a base methodology, and then evaluate each component to determine if its purpose is congruent with the goals of the organization or project. Pikkarainen and Passoja [2005] suggest a similar process. Again, the focus is on organizational goals and how agile methods fit with them. Teams adopt components that are judged effective, while discarding those that do not fit.

Risk-based Methodology

Boehm and Turner [2003] propose a hybrid methodology based on risk assessment (see Figure 3). In this methodology, organizations examine the risk that is present in the project and tailor their life cycle accordingly, using the proposed model to fit needs more fully. Boehm [2010] later expanded this concept to include an incremental commitment model that allows organizations to continually monitor the risk in projects and provides specified points at which a project can be discontinued if risk gets too high.

[2] XP practices selectively adopted include shorter release times, continuous integration, pair programming, simple design, user-driven planning, refactoring, and continuous testing. For an explanation of each of these XP strategies, see Beck and Andres, 2004.

Electronic copy available at: https://ssrn.com/abstract=1909431

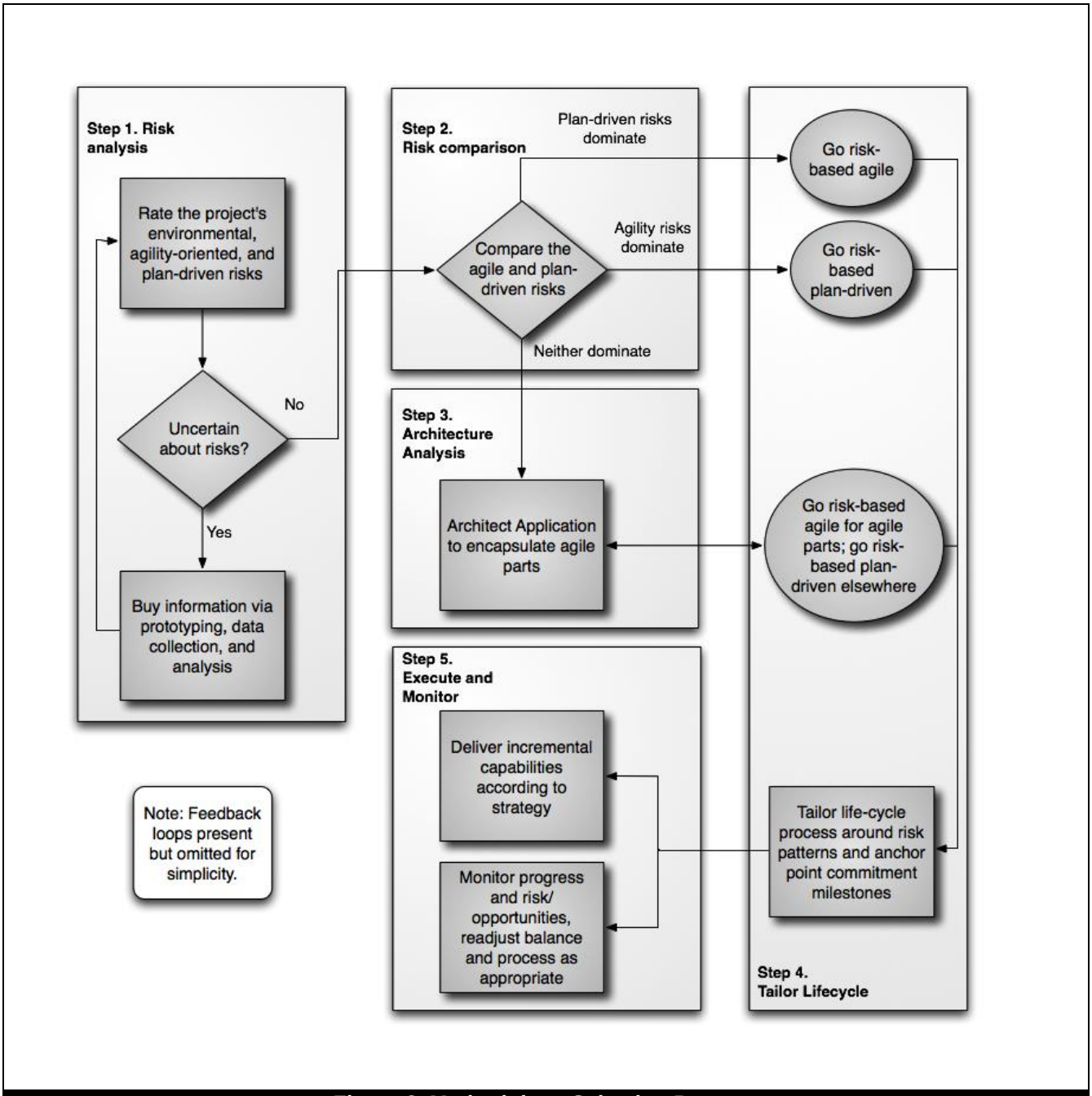


Figure 3. Methodology Selection Process
[Boehm and Turner, "Using Risk to Balance Agile and Plan-Driven Methods," *IEEE Computer,* 2003, p. 60 (© 2003 IEEE) Used with permission]

Cost/Benefit Analysis

Austin and Devin [2009] suggest using a comparison of benefits and costs of specific agile practices and implementing only those agile principles with a greater overall benefit than cost. Using a process similar to the risk-based methodology, the organization evaluates a list of possible agile methods and sums the net benefit of using agile methods in that particular case.

Agile Principles During Only Part of the Life Cycle

Another popular hybrid technique that organizations consider is to use selected practices from agile methodologies while maintaining an overall plan-based development life cycle. For example, a manager could choose to use agile methods such as team-owned decisions and frequent iterations during coding and testing phases [Ambler, 2008]. Projects would still include a full design phase while allowing programmers to enjoy benefits such as frequent feedback and power to make decisions during coding and testing phases.

Electronic copy available at: https://ssrn.com/abstract=1909431

Certain practices of agile development, such as pair programming, can be implemented without interrupting current processes. Research shows that pair programming improves production [Bowers et al., 2002] and reduces code defect density [Fitzgerald et al., 2006]. However, managers should avoid certain agile practices, such as reducing documentation, that do not fit well with an organization's size and culture [Selic, 2009]. Table 4 includes a list of hybrid practices that are particularly suited for large projects and organizations. We adapted the table from research by Cao et al. [2004], who studied a successfully implemented hybrid approach in a large organization. We recommend that large, mature organizations start by considering these practices when designing a hybrid solution using the methods we previously mentioned.

Table 4: Hybrid Practices for Complex, Large-Scale Projects [Cao et al., 2004]

Hybrid Practices Suited for Large Organizations and Projects	Description
Designing upfront	While agile methodologies usually eliminate upfront design, large or complex projects cannot live without some design being done before work begins. For smaller projects, this issue might not be as important.
Short release cycles with layered approach	No matter the size of the project or organization, it is useful to have working software at the end of each cycle to be ready for testing and feedback.
Surrogate customer engagement	Because of the difficulty in soliciting constant feedback from all affected customers on large and complex projects, projects should have product managers who have direct contact with customers for constant feedback on projects.
Flexible pair programming	Pair programming is successful in a wide variety of projects and organizations. Cao et al. [2004] recommend "flexible" pair programming, meaning that developers can use it as much as possible but should be flexible enough to realize that it won't work in all situations.
Identifying and managing developers	Hire and use developers that are more capable of working in an agile development environment.[3] If using hybrid methodologies on only some projects, be sure to assign the right developers.
Reuse with refactoring	Reuse existing code to create new features. While lack of documentation in agile methodologies makes reuse more difficult, refactoring (cleaning up the code so it is easier to adapt to new projects) can help.
Flatter hierarchies with controlled empowerment	Empowering developers to make important decisions in the code makes development faster, and short cycles help to correct any problems that might arise due to this practice.

Service-Oriented Software Development

Another hybrid method, called *Service-Oriented Software Development* (SOSD), consists of dividing the work of a specific project into individual components called *services* [Keith et al., 2009]. The method takes its name from Service-Oriented Architecture (SOA), in which developers create applications from a collection of loosely coupled, independent software services. In the SOSD methodology, sub-teams within the overall project team act as service providers performing independent tasks in the software development process. Interfaces between services or activities are explicitly defined, but the providers of one service do not need to understand the inner workings of any other service. As a result, sub-teams can perform each service required by the overall project plan using their own unique methodology, whether plan-based or agile.

For example, a typical project has design, code, test, and deploy phases. Teams can divide each phase into distinct services to be performed by individuals or small groups. Within the services, sub-services exist to provide specific functionality. One type of sub-service for the development phase would be application development with a database component. Another would be an application component with no database.

When a project is in its planning stages, the team can select and code needed services. The project managers can then map available resources from the organization to the project services. In this way, project planners can easily see the resources available to meet their needs. Although the overall process of coordinating service providers' individual efforts is formal and plan-driven, each unique service can be executed using the methodology of the service provider's choice, including agile methods.

[3] One example, based on Turk et al., 2005, is that agile team members need to be those whose personalities, demographics, skills, etc. will enable them to become well-connected or embedded in the agile team's informal advice network.

Electronic copy available at: https://ssrn.com/abstract=1909431

In summary, the hybrid approaches we highlight provide an appropriate alternative for organizations in which projects are often large and complex. The risk-mitigating features of traditional methods merge with the iterative, customer-focused, more flexible features of agile methods, and strengths of both approaches emerge. In merging approaches, however, some of the risks otherwise mitigated by traditional methods can surface more easily. Organizations must carefully select the best approach that provides a good balance between benefits of traditional and agile, given their risk tolerance and the goals of the project.

Implementation Recommendations

In addition to selecting or developing the right hybrid methodology, the actual implementation of a hybrid methodology requires significant attention. As with any other organizational change in policy or processes, careful change management techniques are necessary in order to switch from plan-based approaches to agile approaches or even hybrid methodologies. Regarding the implementation of hybrid development methodologies, we have several specific recommendations.

Based on our theoretical framework and a review of relevant literature, we first identify two essential requirements for implementing a hybrid solution. First, project managers must recognize the level of, and differences between, interdependencies among project tasks. Doing so allows the appropriate selection of agile vs. plan-based methodologies to coordinate interdependencies for each software process module. Second, to make the first objective possible, managers should accurately modularize the project process into a set of finely granular independent tasks. Teams should separate these tasks along natural breakpoints that also accurately separate the high- versus low-risk tasks and the reciprocal versus sequential interdependency tasks. From our review of existing techniques, these objectives might be accomplished, for example, by adopting Boehm's [2010] risk-based model for the former and Keith et al.'s [2009] SOSD framework for the latter.

In addition, organizations can use pilot tests to determine the effectiveness of hybrid techniques within the organization with reduced risk. Developers should use pilot tests with small projects specifically selected to fit the criteria for hybrid methods as shown in the framework. Selecting well-suited projects will contribute to project success and ensure that hybrid methods are as effective as possible. These pilot tests give insights concerning which practices work and which do not. Also, teams can apply lessons learned during pilot tests when larger portions of the organization are implementing agile methods. Further, organizations need to change some business processes to reflect the agile development process. For example, traditional performance reviews are ineffective when programming is done in pairs [Bowers et al., 2002]. Organizations should adapt policies and processes during pilot testing to measure the effectiveness of the change. In addition, managers should create teams of various levels of experience and expertise. However, we do not recommend including programming novices on the pilot teams [Boehm and Turner, 2005].

Next, organizations should identify specific responsibilities or project types to address with agile methods. Boehm and Turner [2005] suggest using "small, GUI-intensive applications with short life cycles" (p. 32) to begin experimenting with agile techniques within an organization. By limiting the use of agile methods to the right type of projects as specified within our framework, agile methods are more likely to succeed.

The movement from plan-based to agile hybrid methodologies is far from trivial. As our theory and discussion show, having the correct mechanisms in place in an organization to address the volatility and interdependencies of the situation is key. The change will affect all parts of the development process, from project managers and developers to customers. As with any change effort, detractors will arise. Some people might be apprehensive to work in an agile environment. One of the benefits of implementing hybrid methods on select projects is that projects using primarily traditional methods will remain. This benefit should help mitigate some of the antagonism that comes with the change effort.

All parts of the organization that participate in pilots will need to be aware of changes that agile practices bring to the organization. Helping people accept changes can be harder than making the changes [Cockburn and Highsmith, 2001]. Customers need to be more available to consult with project teams. Developers must be more flexible and willing to accept evolving user requirements. Project managers must adapt to new roles when managing an agile team. The role of the project manager could change from one of making decisions to one of facilitating decision-making [McAvoy and Butler, 2009]. In some cases, culture can be so engrained in a project team that some teams might be better off using a methodology that is not necessarily suited to their interdependencies and uncertainties. For example, development teams that are accustomed to clearly defined roles and policies, especially for decision making, should stick primarily to more traditional methods if the culture is such that team members cannot adjust to the change [Boehm and Turner, 2003]. These extreme cases are the exceptions to the framework we present.

Electronic copy available at: https://ssrn.com/abstract=1909431

VII. FUTURE RESEARCH

Our review of literature and theory concerning agile development shows the importance of continued research on the topic. While our theory-based literature review supports several recommendations concerning agile development in large organizations, more empirical research is needed to test these recommendations and to further explore the causes and effects of successful agile development processes in mature organizations.

Our theoretical model and recommendations are also based on a reduced set of fundamental factors that influence methodology selection. We recognize that many other factors identified in the existing literature also impact IT project success, such as team culture [Walsham, 2002], top management support [Schmidt et al., 2001], and alignment with organizational strategy [Slaughter et al., 2006]. Future research should demonstrate how these factors also help to shape software methodology choice.

Future research should also explore agile development success as a function of the density of the project team's advice network, moderated by the cost of maintaining informal relationships. Current literature and theory suggest that an organization's support of informal and formal communication affects the outcomes of the type of development life cycles used. Similarly, researchers should examine whether the collation of roles with reciprocal interdependencies can moderate the effect of a poor social network, and whether investments in knowledge management systems for project knowledge can mitigate the impact of high team volatility.

Finally, additional research should empirically test the antecedents of successful agile development. Most research focuses on limited case studies.

VIII. CONCLUSION

Both plan-based and agile methodologies can be effective ways to develop software. Each method has strengths and weaknesses. By combining the two to create a hybrid methodology, development teams can mitigate weaknesses and create a development methodology even more effective than either one alone. An examination of project interdependencies and volitility allows managers to determine the best type of methodology for a given situation. Accordingly, we recommend that large, mature organizations use our framework to select the appropriate methodology for development. Those facing high uncertainty and reciprocal interdependencies in their software projects should implement a hybrid methodology that combines the strengths of their current software development life cycle with complementary agile practices. Hybrid methodologies allow large, mature organizations to enjoy the benefits of agile development in areas where purely agile methodologies have not previously been successful.

REFERENCES

Editor's Note: The following reference list contains hyperlinks to World Wide Web pages. Readers who have the ability to access the Web directly from their word processor or are reading the article on the Web, can gain direct access to these linked references. Readers are warned, however, that:

1. These links existed as of the date of publication but are not guaranteed to be working thereafter.
2. The contents of Web pages may change over time. Where version information is provided in the References, different versions may not contain the information or the conclusions referenced.
3. The author(s) of the Web pages, not AIS, is (are) responsible for the accuracy of their content.
4. The author(s) of this article, not AIS, is (are) responsible for the accuracy of the URL and version information.

Abrahamsson, P., K. Conboy, and X. Wang (2009) "Lots Done, More to Do: The Current State of Agile Systems Development Research", *European Journal of Information Systems* (18)4, pp. 281–284.

Ambler, S.W. (2008) "Scaling Scrum: Meeting Real-World Development Needs", *Dr. Dobb's Journal* (33)5, pp. 52–54.

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

Bardhan, I.R. et al. (2010) "An Interdisciplinary Perspective on IT Services Management and Service Science", *Journal of Management Information Systems* (26)4, pp. 13–64.

Beck, K. (1999) "Embracing Change with Extreme Programming", *IEEE Computer* (32)10, pp. 70–77.

Beck, K. and C. Andres (2004) *Extreme Programming Explained: Embrace Change, 2nd edition*, Boston, MA: Addison-Wesley Professional.

Electronic copy available at: https://ssrn.com/abstract=1909431

Beck, K. et al. (2001) "Agile Manifesto: Manifesto for Agile Software Development", Agile Alliance, http://agilemanifesto.org (current Feb. 8, 2011).

Berger, H. and P. Beynon-Davies (2009) "The Utility of Rapid Application Development in Large-Scale, Complex Projects", *Information Systems Journal* (19)6, pp. 549–570.

Boehm, B. and R. Turner (2003) "Using Risk to Balance Agile and Plan-Driven Methods", *IEEE Computer* (36)6, pp. 57–66.

Boehm, B. and R. Turner (2005) "Management Challenges to Implementing Agile Processes in Traditional Development Organizations", *IEEE Software* (22)5, pp. 30–39.

Boehm, B.W. (2010) "A Risk-Driven Decision Table for Software Process Selection", *Proceedings of the 2010 International Conference on New Modeling Concepts for Today's Software Processes: Software Process,* Paderborn, Germany, p. 1.

Bowers, J. et al. (2002) "Tailoring XP for Large Mission Critical Software Development", *Proceedings of the Second XP Universe and First Agile Universe Conference,* Chicago, IL, pp. 100–111.

Braithwaite, K. and T. Joyce. (2005) "XP Expanded: Distributed Extreme Programming", *Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering,* Sheffield, UK, pp. 180–188.

Cao, L., K. Mohan, and B. Ramesh. (2004) "How Extreme Does Extreme Programming Have to Be? Adapting XP Practices to Large-Scale Projects", *Proceedings of the 37th Hawaii International Conference on System Sciences,* Waikoloa, Hawaii, pp. 1–10.

Cao, L. et al. (2009) "A Framework for Adapting Agile Development Methodologies", *European Journal of Information Systems* (18)4, pp. 332–343.

Cockburn, A. and J. Highsmith (2001) "Agile Software Development: The People Factor", *IEEE Computer* (34)11, pp. 131–133.

Cohn, M. (2007) "Advice on Conducting the Scrum of Scrums Meeting", Scrum Alliance, http://www.scrumalliance.org/articles/46-advice-on-conducting-the-scrum-of-scrums-meeting (current Feb. 8, 2011).

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

Dennis, A.R., R.M. Fuller, and J.S. Valacich (2008) "Media, Tasks, and Communication Processes: A Theory of Media Synchronicity", *MIS Quarterly* (32)3, pp. 575–600.

Donaldson, L. (2001) *The Contingency Theory of Organizations*, London, England: Sage Publications.

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

Erickson, J., K. Lyytinen, and K. Siau (2005) "Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research", *Journal of Database Management* (16)4, pp. 88–100.

Faraj, S. and S. Lee (2000) "Coordinating Expertise in Software Development Teams", *Management Science* (46)12, pp. 1554–1568.

Fitzgerald, B., G. Hartnett, and K. Conboy (2006) "Customising Agile Methods to Software Practices at Intel Shannon", *European Journal of Information Systems* (15)2, pp. 200–213.

Galbraith, J.R. (1973) *Designing Complex Organizations*, Reading, MA: Addison-Wesley.

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

Holmstrom, H. et al. (2006) "Agile Practices Reduce Distance in Global Software Development", *Information Systems Management* (23)3, pp. 7–18.

Kahkonen, T. (2004) "Agile Methods for Large Organizations: Building Communities of Practice", *Proceedings of the 2nd Annual Agile Development Conference,* Salt Lake City, UT, pp. 2–10.

Karlsson, F. and P. Agerfalk (2009) "Exploring Agile Values in Method Configuration", *European Journal of Information Systems* (18)4, pp. 300–316.

Keith, M., H. Demirkan, and M. Goul. (2009) "Service-Oriented Software Development", *Proceedings of the 15th Americas Conference on Information Systems,* San Francisco, CA, pp. 1–10.

Electronic copy available at: https://ssrn.com/abstract=1909431

Keith, M., H. Demirkan, and M. Goul (2010) "The Influence of Collaborative Technology Knowledge on Advice Network Structures", *Decision Support Systems* (50)1, pp. 140–151.

Kettunen, P. (2009) "Adopting Key Lessons from Agile Manufacturing to Agile Software Product Development—A Comparative Study", *Technovation* (29)6–7, pp. 408–422.

Klein, K.J. et al. (2004) "How Do They Get There? An Examination of the Antecedents of Centrality in Team Networks", *Academy of Management Journal* (47)6, pp. 952–963.

Kraut, R.E. and L.A. Streeter (1995) "Coordination in Software Development", *Communications of the ACM* (38)3, pp. 69–81.

Kussmaul, C., R. Jack, and B. Sponsler (2004) "Outsourcing and Offshoring with Agility: A Case Study" in Zannier, C., H. Erdogmus, and L. Lindstrom (eds.) *Extreme Programming and Agile Methods—XP/Agile Universe 2004*, Heidelberg, Germany: Springer, pp. 147–154.

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

Lindvall, M. et al. (2004) "Agile Software Development in Large Organizations", *IEEE Computer* (37)12, pp. 26–34.

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

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

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

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

Nonaka, I. (1994) "A Dynamic Theory of Organizational Knowledge Creation", *Organization Science* (5)1, pp. 14–37.

Panzar, J.C. and R.D. Willig (1981) "Economies of Scope", *American Economic Review* (71)2, pp. 268–272.

Pikkarainen, M. et al. (2008) "The Impact of Agile Practices on Communication in Software Development", *Empirical Software Engineering* (13)3, pp. 303–337.

Pikkarainen, M. and U. Passoja. (2005) "An Approach for Assessing Suitability of Agile Solutions: A Case Study", *Proceedings of the 6th International Conference of eXtreme Programming and Agile Process in Software Engineering,* Sheffield University, UK, pp. 171–171.

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

Rising, L. and N.S. Janoff (2000) "The Scrum Software Development Process for Small Teams", *IEEE Software* (17)4, pp. 26–32.

Royce, W.W. (1970) "Managing the Development of Large Software Systems: Concepts and Techniques", *Proceedings of the 9th International Conference on Software Engineering,* Monterey, CA, pp. 328–338.

Sarker, S. (2009) "Exploring Agility in Distributed Information Systems Development Teams: An Interpretive Study in an Offshoring Context", *Information Systems Research* (20)3, pp. 440–461.

Sarker, S., C. Munson, and S. Chakraborty (2009) "Assessing the Relative Contribution of the Facets of Agility to Distributed Systems Development Success: An Analytic Hierarchy Process Approach", *European Journal of Information Systems* (18)4, pp. 285–299.

Schmidt, R. et al. (2001) "Identifying Software Project Risks: An International Delphi Study", *Journal of Management Information Systems* (17)4, pp. 5–36.

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

Selic, B. (2009) "Agile Documentation, Anyone?" *IEEE Software* (26)6, pp. 11–12.

Slaughter, S.A. et al. (2006) "Aligning Software Processes with Strategy", *MIS Quarterly* (30)4, pp. 891–918.

Electronic copy available at: https://ssrn.com/abstract=1909431

Sutherland, J. et al. (2007) "Distributed Scrum: Agile Project Management with Outsourced Development Teams", *Proceedings of the 40th Annual Hawaiian International Conference on Systems Sciences,* Waikoloa, Hawaii, pp. 271–283.

Thompson, J.D. (1967) *Organizations in Action*, New York, NY: McGraw-Hill.

Turk, D., R. France, and B. Rumpe (2005) "Assumptions Underlying Agile Software-Development Processes", *Journal of Database Management* (16)4, pp. 62–87.

Tushman, M.L. and D.A. Nadler (1978) "Information Processing as an Integrating Concept in Organizational Design", *The Academy of Management Review* (3)3, pp. 613–624.

Vidgen, R. and X. Wang (2009) "Coevolving Systems and the Organization of Agile Software Development", *Information Systems Research* (20)3, pp. 355–376.

Walsham, G. (2002) "Cross-Cultural Software Production and Use: A Structurational Analysis", *MIS Quarterly* (26)4, pp. 359–380.

APPENDIX A

Table A-1: Previous Research on Agile Methods

	Experimental	Case Study	Theory-building	Design Science	Review/ Commentary
Agile	None that we are aware of	Berger and Beynon-Davies, 2009 Bowers et al., 2002 Braithwaite and Joyce, 2005 Cao et al., 2004 Cao et al., 2009 Conboy, 2009 Fitzgerald et al., 2006 Kahkonen, 2004 Kussmaul et al., 2004 Lee and Xia, 2010 Lindvall et al., 2004 Mangalaraj et al., 2009 Maruping et al., 2009a Maruping et al., 2009b Pikkarainen and Passoja, 2005 Pikkarainen et al., 2008 Rising and Janoff, 2000 Sarker, 2009 Selic, 2009 Sutherland et al., 2007 Turk et al., 2005	Austin and Devin, 2009 Harris et al., 2009 Sarker, 2009 Vidgen and Wang, 2009	Ambler, 2008 Beck, 1999 Bowers et al., 2002 Braithwaite and Joyce, 2005 Cao et al., 2009 Port and Bui, 2009 Rising and Janoff, 2000] Schwaber and Beedle, 2002	Abrahamsson et al., 2009 Cockburn and Highsmith, 2001 Dyba and Dingsoyr, 2008 Erickson et al., 2005 Holmstrom et al., 2006 Kettunen, 2009
Hybrid	Port and Bui, 2009	Ambler, 2008 Boehm and Turner, 2003 Kahkonen, 2004 Karlsson and Agerfalk, 2009 McAvoy and Butler, 2009	Cao et al., 2009	Keith et al., 2009	Boehm and Turner, 2005

Electronic copy available at: https://ssrn.com/abstract=1909431

ABOUT THE AUTHORS

Jordan B. Barlow is a graduate of the Masters of Information Systems program of the Marriott School of Management of Brigham Young University where he was enrolled in the Information Systems Ph.D. Preparation Program. He will be starting his Ph.D. in Information Systems at Indiana University during Fall 2011. His research interests include virtual communication and collaboration, HCI, knowledge management, trust, and IT security and risk. He has worked in IT training and management at the LDS Missionary Training Center and as a research assistant at BYU. He is currently employed as a small business/technology consultant at Squire & Co., PC, in Orem, Utah.

Justin Scott Giboney is a graduate of the Masters of Information Systems program of the Marriott School of Management of Brigham Young University, where he completed the Information Systems Ph.D. Preparation Program. He is currently working on a Ph.D. in Management Information Systems at the University of Arizona's Eller College of Management. His research interests include trust, expectations in computer interactions, and group collaboration.

Mark Jeffrey Keith is an Assistant Professor in the Computer Information & Decision Management Department at the College of Business at West Texas A&M University. He completed his doctorate in Information Systems at Arizona State University. He received a BS in Information Systems and a Master of Information Systems Management (while enrolled in the Ph.D. Preparation Program) from Brigham Young University. His recent research interests concern IT project effectiveness including the impact of advice networks and service-oriented principles on project success, IT switching costs, and mobile computing. His research has appeared in the *Journal of the Association for Information Systems, Decision Support Systems, Decision Analysis,* and the *International Journal of Human-Computer Studies.*

David W. Wilson is a graduate of the Masters of Information Systems program of the Marriott School of Management of Brigham Young University, where he completed the Information Systems Ph.D. Preparation Program. He is a doctoral student at Washington State University in the Department of Entrepreneurship and Information Systems. His research interests include social computing systems, collaboration, trust, and HCI.

Ryan M. Schuetzler is a graduate of the Masters of Information Systems program of the Marriott School of Management of Brigham Young University, where he completed the Information Systems Ph.D. Preparation Program. He is currently working on a Ph.D. in Management Information Systems at the University of Arizona's Eller College of Management. His research interests include trust and distrust, computer-mediated collaboration, and deception detection.

Paul Benjamin Lowry is an Associate Professor of Information Systems at City University of Hong Kong. His research interests include Human-Computer Interaction (HCI) (collaboration, culture, communication, adoption, entertainment, decision support, deception), e-commerce (privacy, security, trust, branding, electronic markets), and scientometrics of Information Systems research. He received his Ph.D. in Management Information Systems from the University of Arizona. He published articles in *MIS Quarterly*; *Journal of Management Information Systems*; *Journal of the Association for Information Systems*; *Information Systems Journal*; *European Journal of Information Systems*; *Communications of the ACM*; *Information Sciences*; *Decision Support Systems*; *IEEE Transactions on Systems, Man, and Cybernetics*; *IEEE Transactions on Professional Communication*; *Small Group Research*; *Expert Systems with Applications*; *Communications of the Association for Information Systems*; and others. He currently serves as a guest associate editor for *MIS Quarterly*, and regularly appointed associate editor at *European Journal of IS*, *Electronic Commerce Research and Applications*, *Communications of the Association for Information Systems*, and *AIS Transactions on HCI*.

Anthony Vance is an Assistant Professor of Information Systems in the Marriott School of Management of Brigham Young University. He has earned Ph.D. degrees in Information Systems from Georgia State University, US; the University of Paris–Dauphine, France; and the University of Oulu, Finland. He received a B.S. in IS and Masters of Information Systems Management (MISM) from Brigham Young University, during which he was enrolled in the IS Ph.D. preparation program. His previous experience includes working as a visiting research professor in the Information Systems Security Research Center at the University of Oulu, where he remains a research fellow. He also worked as an information security consultant for Deloitte. His work is published in *MIS Quarterly*, *Journal of Management Information Systems*, and *European Journal of Information Systems*. His research interests are information security, trust in information systems, and internal control.

Electronic copy available at: https://ssrn.com/abstract=1909431

Copyright © 2011 by the Association for Information Systems. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation on the first page. Copyright for components of this work owned by others than the Association for Information Systems must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or fee. Request permission to publish from: AIS Administrative Office, P.O. Box 2712 Atlanta, GA, 30301-2712, Attn: Reprints; or via e-mail from ais@aisnet.org.

Electronic copy available at: https://ssrn.com/abstract=1909431

Communications of the Association for Information Systems

ISSN: 1529-3181

EDITOR-IN-CHIEF
Ilze Zigurs
University of Nebraska at Omaha

AIS PUBLICATIONS COMMITTEE

Kalle Lyytinen Vice President Publications Case Western Reserve University	Ilze Zigurs Editor, *CAIS* University of Nebraska at Omaha	Shirley Gregor Editor, *JAIS* The Australian National University
Robert Zmud AIS Region 1 Representative University of Oklahoma	Phillip Ein-Dor AIS Region 2 Representative Tel-Aviv University	Bernard Tan AIS Region 3 Representative National University of Singapore

CAIS ADVISORY BOARD

Gordon Davis University of Minnesota	Ken Kraemer University of California at Irvine	M. Lynne Markus Bentley University	Richard Mason Southern Methodist University
Jay Nunamaker University of Arizona	Henk Sol University of Groningen	Ralph Sprague University of Hawaii	Hugh J. Watson University of Georgia

CAIS SENIOR EDITORS

Steve Alter University of San Francisco	Jane Fedorowicz Bentley University	Jerry Luftman Stevens Institute of Technology

CAIS EDITORIAL BOARD

Monica Adya Marquette University	Michel Avital University of Amsterdam	Dinesh Batra Florida International University	Indranil Bose University of Hong Kong
Thomas Case Georgia Southern University	Evan Duggan University of the West Indies	Mary Granger George Washington University	Åke Gronlund University of Umea
Douglas Havelka Miami University	K.D. Joshi Washington State University	Michel Kalika University of Paris Dauphine	Karlheinz Kautz Copenhagen Business School
Julie Kendall Rutgers University	Nancy Lankton Marshall University	Claudia Loebbecke University of Cologne	Paul Benjamin Lowry City University of Hong Kong
Sal March Vanderbilt University	Don McCubbrey University of Denver	Fred Niederman St. Louis University	Shan Ling Pan National University of Singapore
Katia Passerini New Jersey Institute of Technology	Jan Recker Queensland University of Technology	Jackie Rees Purdue University	Raj Sharman State University of New York at Buffalo
Mikko Siponen University of Oulu	Thompson Teo National University of Singapore	Chelley Vician University of St. Thomas	Padmal Vitharana Syracuse University
Rolf Wigand University of Arkansas, Little Rock	Fons Wijnhoven University of Twente	Vance Wilson Worcester Polytechnic Institute	Yajiong Xue East Carolina University

DEPARTMENTS

Information Systems and Healthcare Editor: Vance Wilson	Information Technology and Systems Editors: Sal March and Dinesh Batra	Papers in French Editor: Michel Kalika

ADMINISTRATIVE PERSONNEL

James P. Tinsley AIS Executive Director	Vipin Arora CAIS Managing Editor University of Nebraska at Omaha	Sheri Hronek CAIS Publications Editor Hronek Associates, Inc.	Copyediting by S4Carlisle Publishing Services

Electronic copy available at: https://ssrn.com/abstract=1909431