Communications of the Association for Information Systems

Volume 36 Article 5

2-2015

Running on Hybrid: Control Changes when Introducing an Agile Methodology in a Traditional "Waterfall" System Development Environment

Lakshman Mahadevan
Emporia State University, lmahadev@emporia.edu

William J. Kettinger
University of Memphis

Thomas O. Meservy
Brigham Young University

Recommended Citation
Mahadevan, Lakshman; Kettinger, William J.; and Meservy, Thomas O. (2015) "Running on Hybrid: Control Changes when Introducing an Agile Methodology in a Traditional "Waterfall" System Development Environment," *Communications of the Association for Information Systems*: Vol. 36 , Article 5.
DOI: 10.17705/1CAIS.03605
Available at: https://aisel.aisnet.org/cais/vol36/iss1/5

Communications of the Association for Information Systems

CAIS

Running on Hybrid: Control Changes when Introducing an Agile Methodology in a Traditional "Waterfall" System Development Environment

Lakshman Mahadevan
Emporia State University
lmahadev@emporia.edu

William J. Kettinger
University of Memphis

Thomas O. Meservy
Brigham Young University

Abstract:

Prior to implementing "Agile" software development methods, organizations rooted in traditional "Waterfall" software development employed heavy upfront project design and limited changes and feedback during and between project stages. Waterfall methods make heavy use of outcome controls primarily monitored by the information systems function (ISF). This paper explores the control mechanisms used by the ISF and business function (BF) during and after the introduction of a major Agile project at a large U.S. company steeped in the traditional Waterfall method. Outcome control, the predominant control mechanism used in the case company, gave way to a hybrid-like control that possessed mechanisms of emergent control while maintaining vestiges of some Waterfall-like outcome control. We observed that, prior to the introduction of Agile, the software-development process was firmly in the hands of the ISF. The introduction of Agile shifted some of the controller authority over the development process from the ISF to the BF. Lessons learned from the case study point to the complexity of designing control mechanisms during a transition from the Waterfall method to an Agile approach.

Keywords: Agile, Software-Development Process, Hybrid Control, Waterfall, SDLC, Business Function, Information Systems Function, Clan Control, Self-Control, Outcome Control, Behavioral Control, Emergent Control.

Volume 36, Article 5, pp. 77-103, February 2015

The manuscript was received 23/07/2012 and was with the authors 12 months for 2 revisions.

I. INTRODUCTION

The traditional "Waterfall" software-development approach is a downward-flowing stage model for developing software requiring substantial upfront design. Feedback is limited between stages of the system development lifecycle (SDLC), including specifications, development, testing, and implementation (Boehm, 1988). The Waterfall approach rose to prominence in the 1970s in the highly structured aerospace and manufacturing industries. In these industries, after-the-fact changes are prohibitively expensive. The Waterfall approach grew to be a dominant software-development methodology in many large companies. However, proponents of the Agile software-development method argue the Waterfall method is flawed because it is almost impossible for any non-trivial project to finish a SDLC phase completely as pre-specified. Agile proponents claim that changes and learning must take place throughout a project. As Agile has grown in favor, large companies are increasingly transitioning from Waterfall to Agile methods.

Transition is not easy. The Agile approach is very different from the Waterfall approach in several ways. In the Waterfall approach, responsibility is often housed in the information systems function (ISF). In the Agile approach, joint project responsibility is assigned to the ISF and business function (BF) areas. During the project, representatives from both functions are co-located. Agile team members jointly provide status reports on a daily basis. Iteration cycles are only a few weeks long and involve customer and management feedback at the end of each session. Requirements are constantly evaluated and feature priorities are either upgraded or downgraded depending on customer intervention. Joint responsibility, daily reporting, multiple quick iterations, and volatility in requirements make a switch to the Agile methodology a significant organizational transition.

Control relationships play an important role in helping organizations achieve their objectives. Switching to a new Agile software-development methodology can change the control relationships between departments. Switching can also disrupt the coordination activities between the controller (manager) and controlee (contributor). Research shows that controllers in the ISF and the BF use formal controls to complete objectives, while controlees tend to use more informal controls to achieve their objectives[1]. Compared to the Waterfall approach, the Agile approach is more group intensive. It involves almost daily interaction between the ISF's members and the BF contributors. The Agile methodology tends to be less-structured, with more unspecified outcomes. Project control dynamics in the Agile development method place a much greater role on clans and self-control (McHugh, Conboy, & Lang, 2011) than is common in the Waterfall method. Thus, for an organization introducing Agile methods, a change in development approach that relies more on individuals or teams to exercise control presents important resource allocation and control consequences.

While previous research in Agile development has looked at control relationships, most studies limit their focus to the ISF's boundaries (Harris, Collins, & Hevner, 2009). Also, previous research does not explicitly recognize changes to control mechanisms when Agile was introduced in an organization deeply rooted in the Waterfall methodology. In order to advise companies as they transition to Agile, we require better understanding of the dynamics of control. Without such understanding, organizations may not appreciate what they are getting into in adopting Agile, which could limit potential benefits and possibly bias cost benefit assessments.

We are mindful of the coordination challenges involved in integrating Agile practices with existing Waterfall standards and business processes. We evaluated the introduction of the Agile method in an organization well entrenched in Waterfall methods, and conducted a detailed case analysis of the interaction between inter-functional contributors using control theory (Barlow et al., 2011). We focused on the following research question: How does the introduction of the Agile method in a traditional Waterfall-oriented software-development environment change the controller-controlee relationship in projects and across functional boundaries?

We selected a specific Fortune 100 company based on a success story conveyed by two of its senior vice presidents (one from ISF and the other from BF) in a dynamic, coordinated presentation. Both senior vice presidents jointly touted the benefits of Agile and its effective implementation. The company was experienced in Waterfall

[1] In the SDLC, if a project manager is more knowledgeable about the software-development process, behavioral controls that track each step are more likely to be employed, whereas a lack of detailed knowledge of the process suggests the use of outcome controls. In this case, the finished product is the object of scrutiny (see Kirsch, 1996).

software-development methodologies and had piloted a project using Agile methodologies. The senior vice presidents suggested that their Agile implementation was successful and would serve as an exemplar going forward for future development projects in the organization.

II. LITERATURE

IS Research on Agile Development

A growing body of academic literature regarding Agile methodologies and practices contributes to our understanding of how and when Agile methodologies can be applied (Dyba & Dingsoyr, 2008). Researchers have investigated how to tailor Agile techniques in specific environments (Fitzgerald, Hartnett, & Conboy, 2006). The similarities and differences between Agile techniques and Waterfall development methods (McAvoy & Butler, 2007; Nerur, Mahapatra, & Mangalaraj, 2005) have been researched. Environments and contexts for which Agile techniques are best suited (Boehm & Turner, 2003; Cockburn & Highsmith, 2001) have been reviewed, and challenges associated with implementing Agile software development (Nerur et al., 2005) investigated.

Agile development is suitable when there is a high degree of uncertainty and risk in a project that arises from frequently changing requirements and/or the novelty of technology used (Boehm & Turner, 2003, Cockburn & Highsmith, 2001). Researchers have conceptualized how Agile and Waterfall development techniques might co-exist in the same organization (Vinekar, Slinkman, & Nerur, 2006). They have looked at combining specific Agile methods with conventional project management techniques to improve software quality, communication, and product functionality (Karlström & Runeson, 2005). Agile adoption usually takes place from the bottom up in small development teams championed by a low number of highly effective people. Despite initial success at the team level, some companies find it difficult to implement Agile beyond specific projects (Abrahamsson, Conboy, & Wang, 2009) and to integrate Agile into Waterfall-oriented top-down systems development organizations (Boehm & Turner, 2005).

Control in Software Development

Controls are mechanisms that allow an organization to move toward its objectives. Controls are focused on those that exhibit control (i.e., the controller) and those that are influenced or controlled (i.e., controlee) (Ouchi, 1979). Formal and informal controls are two broad categories of control discussed in the literature (Jaworski, 1988; Ouchi, 1979). Two types of formal controls are behavior-based controls and outcome-based controls. Examples of informal controls include clan control and self-control. Behavior control is appropriate when there is near-perfect knowledge of the input-to-output transformation process. Controllers define appropriate steps and procedures for task performance and evaluate controlees' performance based on their prescribed procedures (Kirsch, Sambamurthy, Ko, & Purvis, 2002). Outcome control is exercised when there is imperfect knowledge or no knowledge of the transformation process and only the process's output can be measured. The controller evaluates the controlee on whether the outcomes were met and not on the process used to achieve the targets (Kirsch, 1996). Clan control is appropriate where neither the behaviors nor outputs can be measured properly. Members of a clan belong to a common organization and share values, beliefs, and attitudes. The clan ensures that members behave appropriately (Kirsch, 1997; Kohli & Kettinger, 2004). Self-control is exercised when an individual monitors their own behavior and rewards or sanctions themselves (Henderson & Lee, 1992; Kirsch, 1996). Organizations assume that the individual makes choices relative to the values or the objectives of the organization rather than personal values or objectives.

Research into the use of control structures in ISF is limited to projects that use a plan-driven approach such as the Waterfall approach (McHugh, Kieran, & Michael, 2008). This involves creating a priori specifications of the requirements, quality metrics, budgets, and schedules early in the development process. After development is completed, the delivered output is checked for compliance with the a priori specification. In contrast, the Agile development process focuses on individual interactions over process and tools, on customer interaction throughout the development process, and on changeable requirements.

To gain broad exposure, Agile provides feedback mechanisms through daily team review meetings, co-location of team members and customer representatives, pair programming, and very short release cycles. Harris et al. (2009) note that outcome control used in the Waterfall context is not suited for the Agile development process since the latter's outcomes emerge from a less-structured iterative-development process. Control using Waterfall requires concrete plans and processes. In Agile, the outcome is only known when the process is complete (Harris et al., 2009). The lack of pre-set specifications at the outset of the Agile development process and the evolving nature of the requirements provide a case for using emergent control mechanisms. While outcome control evaluates the final output, emergent control steers the output's evolution (Harris et al., 2009). The ability to continuously demonstrate the software as it emerges from development allows the team to more rapidly adjust its direction. In control theory

terminology, the Agile methodology delivers emergent software through various iterations and is quite different from outcome control (see Table 1).

Table 1: Outcome Control vs. Emergent Control (Adapted from Harris et al., 2009).

Control	Purpose	Frequency	Evaluator	Construction of standard	Comparison
Outcome control	Evaluation	Once	Manager	A priori	Completed projected versus specification
Emergent control	Corrective action	Continuous	Multiple stakeholders	Evolving by stakeholder	Emergent outcomes versus tacit specifications

In practice, the difference between Agile and Waterfall may not be as stark as those in Table 1. Many large organizations find it necessary to run multiple software-developmental regimes, whereby Waterfall projects can run concurrently with Agile projects. While Harris et al., 2009 recognize that Agile encourages emergent controls, Agile projects in companies that operate with multiple software-developmental regimes may take on more hybrid control approaches. In a hybrid approach, the specific Agile approach used possesses some attributes of the Waterfall approach (Barlow et al., 2011) and some of emergent control. For example, in a hybrid context, researchers have observed that more-detailed documentation may be necessary (Bose, 2008; Cao, Mohan, Xu, & Ramesh, 2009) than is typical in a purely Agile approach. In hybrid control, documentation establishes measurable expectation levels at the beginning of a sprint cycle. The expectation levels are evaluated at the sprint's end. Hybrid control is the middle ground between structured a priori control mechanisms used in the Waterfall approach and less-structured, more-fluid emergent control mechanisms primarily used in pure Agile-development scenarios. In addition to emergent control, hybrid control (i.e., blended aspects of emergent and outcome control) may be employed during an Agile-development project. With our case study, we further explore in detail how hybrid control takes shape during the different phases of the Agile approach.

III. CASE STUDY

We collected data from an organization based in the mid-south region of the US. The organization was comprised of more than 280,000 employees worldwide, with more than 10,000 employees in the IT division. The company exceeded 50 development divisions and used the Waterfall approach for software development. The results address controller-controlee relationships in different organizational units during Agile's introduction.

The organization's e-commerce area supported customer applications for various requirements over the Internet such as order tracking, address verification, and rate information using both online and offline modes. Studies by the organization's marketing division showed that the organization's service offerings did not present an integrated picture of the organization to customers. Various flavors of the same service existed to target varying customer types and business segments with unique requirements. The organization launched a new initiative that focused on delivering a single integrated customer experience across all the organization's applications. The organization's goal was to retain and broaden its base of customers by offering the best breed of automated solutions across business segments and customer types. The end state offered customers a unified, seamless, digital access experience that made their reporting, tracking, managing, administering, and printing simple and easy whether online or offline. The Agile team was tasked with streamlining the "becoming a customer" process by simplifying user registration, account setup, and logged-in state.

Previously, the online account and discount registration application was separate from the login application. The customer had to provide the information twice to the organization. The target segment included all new customers who wanted a login to the organization's Internet website. They also targeted only the US and Canada and identified only two business segments for the initial rollout. The Agile team needed to deliver 1) visibility to registration fields, 2) simplified billing processes using the already-received registrant information, 3) prompts to change billing information when changing contact information, 4) evidence that registration information was validated, and 5) emails that detailed successful registration.

Business function representatives approached the corporate software-development process's governance body to request they use Agile methods. The BF team was motivated by experiences of other companies reported in the popular press and their discussions with Agile methodology practitioners. Moreover, the team was inspired by the ability of Agile techniques to reduce defects and to improve a final product's quality. As the ISF and BF were transitioning from the Waterfall approach to using the Agile methodology, we observed that the team implemented a home-grown Agile hybrid solution that followed the principles laid out by Agile's Scrum methodology but also allowed

for outcome reporting using the company's traditional Waterfall method. In essence, a hybrid approach was followed.

Team Setup

The Agile team was split into two groups: the core team and the Scrum team (see Figure 1). The Scrum team consisted of a Scrum master, two product owners (one responsible for account management and one for the login application), and four developers (two associated with each of the applications). These two product owners were responsible for understanding, documenting, and approving the changes that would impact their respective product lines. A test lead was also a member of the Scrum team. The core team consisted of the core team lead, the marketing/sales, finance, and IT lead, and the customer-support lead. The Scrum team test lead was also a member of the core team. Four ISF and three BF managers had to partially relinquish their personnel resources to create the Scrum and core team.

The core team reported progress about the Scrum team to the organization's governance body. The core team also shielded the Scrum team from process and governance issues, and solved organizational and managerial issues that came up. The Scrum team developed the software based on iterations and Scrum methods. The Agile team (Scrum and core) picked the Scrum process as their preferred method of developing software based on discussions with consultants and references in practitioner journals.

The Scrum team was responsible for planning, developing, and testing the software product. The team constantly assessed the software's quality and progress following the implementation of each of the requested features and quickly resolved any discrepancies, sometimes as early as the next day.

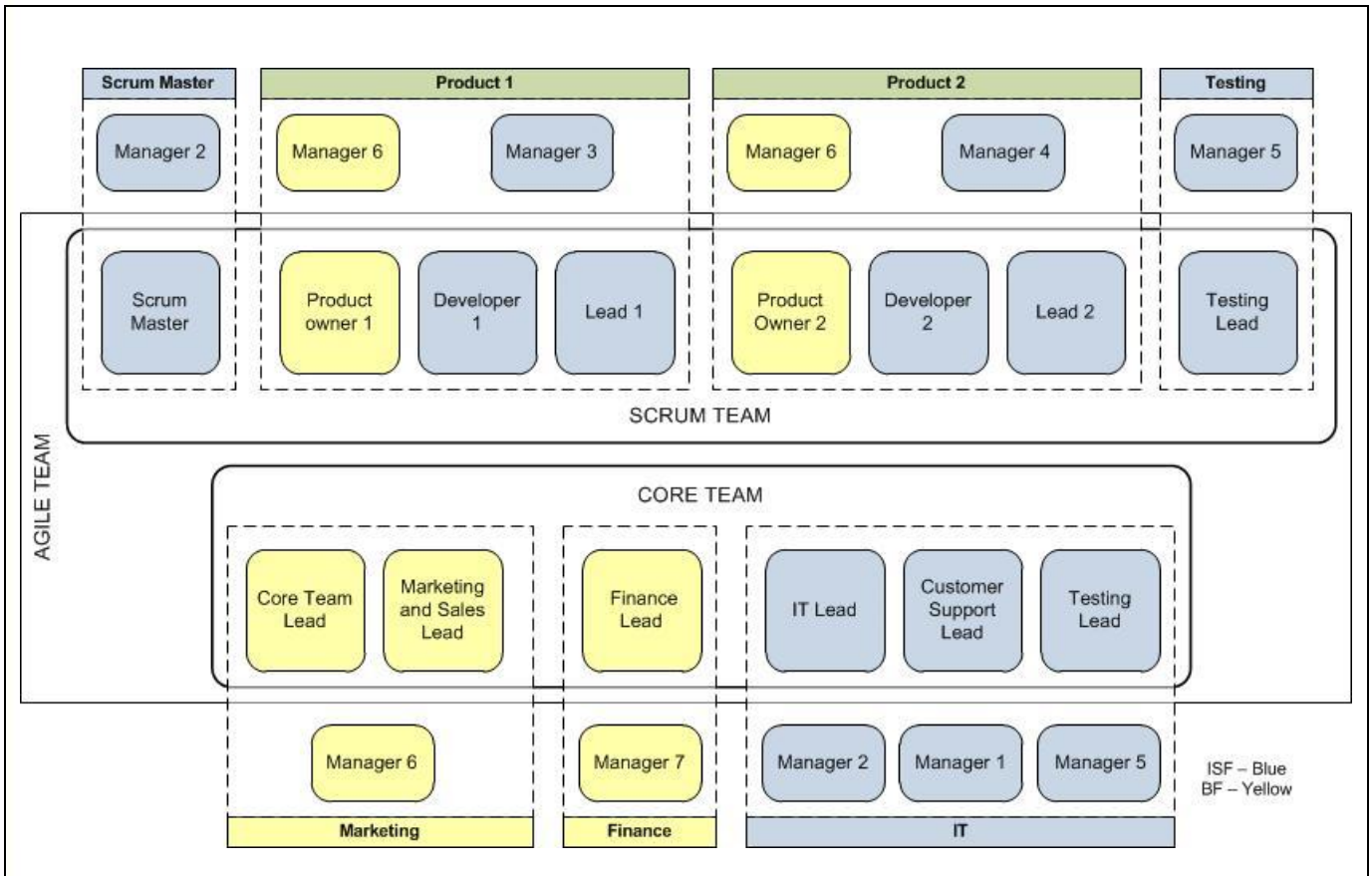


Figure 1. Agile Team Composition including Scrum Participants and Core Team Members

The introduction of an Agile software development methodology, which, in large organizations, requires individuals across all levels to coordinate and collaborate, presents an interesting yet complex and fertile environment to study the control relationships between the organization's functions. Figure 2 shows a subset of the organizational hierarchy as it related to the Agile team. Three executive vice president's (EVP) were involved in the strategic direction of the Agile development project. The ISF reported to three senior vice presidents (SVP), while the BF reported through two SVP to their respective EVP. In the ISF, we can see that two teams, a development team and

a requirements analysis team, reported to one vice president (VP). The Agile team was comprised of team members working laterally across five SVPs' and three EVPs' organizations.

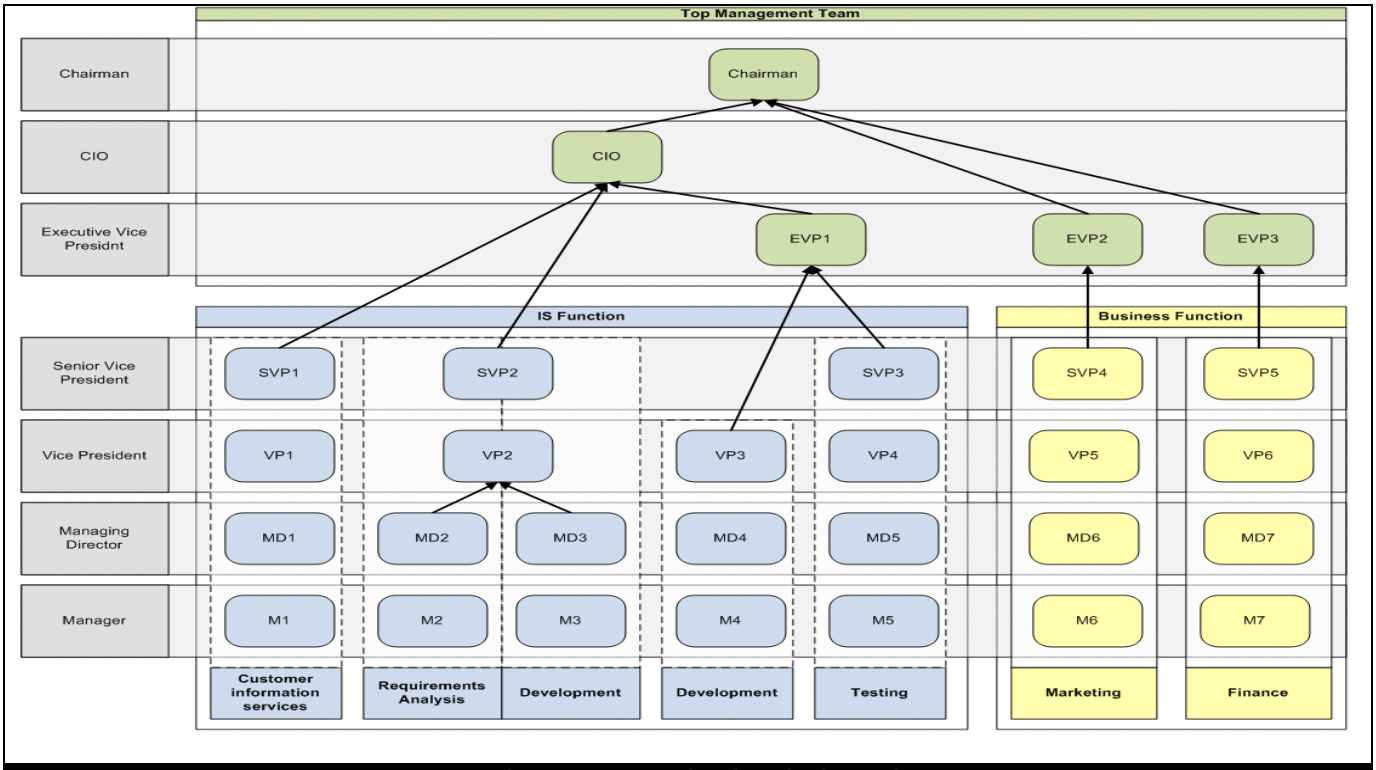


Figure 2. Organizational Hierarchy

Methods of Data Collection and Analysis

We used qualitative case study research methodology to describe and analyze the introduction of Agile in the studied organization. Case study research is a well-known methodology (Eisenhardt, 1985; Yin, 2009). Our case provided an opportunity to observe and analyze an understudied phenomenon that is generally inaccessible to scientific investigation (Yin, 2009). The inter-functional impacts of Agile development when introduced in an organization that is transitioning from embedded use of the traditional Waterfall approach has been understudied thus far and is, therefore, a good application of the case study methodology.

Our data collection methods included interviews, document reviews, focus groups, and follow-up emails. We conducted interviews one-on-one with participants; these interviews were based on five or six open-ended questions derived from the main research question. We analyzed interview transcripts and focus group audio, and used a systematic approach to extract topics related to our research question. We also used archival materials, including presentations that the Agile team had presented and examples of burn-down charts and project timeline estimates, to understand the Agile approach's introduction.

Participants interviewed during the study included two members of the development teams who were also active members of the Scrum team (see Figure 2). We interviewed the Scrum master and one information system function manager, one tester who was a member of the Scrum and the core team, one BF contributor who was a member of the core team, and one BF manager who was responsible for the Agile team. We selected the participants based on the Scrum master's (the principal contact in the organization) recommendations.

The Scrum Process

Although the Agile team employed underlying Scrum principles, they focused on principles that allowed them to complete their work quickly while maintaining high-quality standard of software production. They developed their Agile hybrid solution themselves and did not purely adopt from Agile implementations by other mature organizations in the industry. The Scrum team's members were accountable to each other. In the daily standup meeting, each team member responsible for a piece of work was asked to report the status in order to ensure accountability. In addition to the Scrum team, there were two specific roles that were used to guide the development process: the product owner and the Scrum master. The product owner represented the business's/customer's interests in the application. The product owner for the application was the marketing manager (manager 6). The Scrum master was

charged with helping the Agile team perform to its highest ability using the Scrum methodology. The Scrum master was akin to an orchestra conductor: he brought together a group of talented people and directed them in a way that produced, as it were, wonderful music.

The initiative's primary goal was rapid delivery of high-quality software. Fewer defects meant high quality, rapidly produced software. Defects detected earlier in the development lifecycle are much less expensive than those detected later in the process lifecycle. The Agile team viewed the Scrum process as repeatable, and mapped ito the Waterfall software-development process used in the organization.

The Agile team replaced industry-accepted terminology with new terms for the Scrum process used by the organization so that its governance body could understand them. The organization replaced terms such as sprint backlog and sprint planning with terms such as product backlog and release planning. They included the sprint backlog, or stories worked in a particular iteration, as part of iteration planning.

Figure 3 captures the main phases of the Waterfall processes used at the case site. The figure also shows the relation of these phases to major activities in the Scrum process.

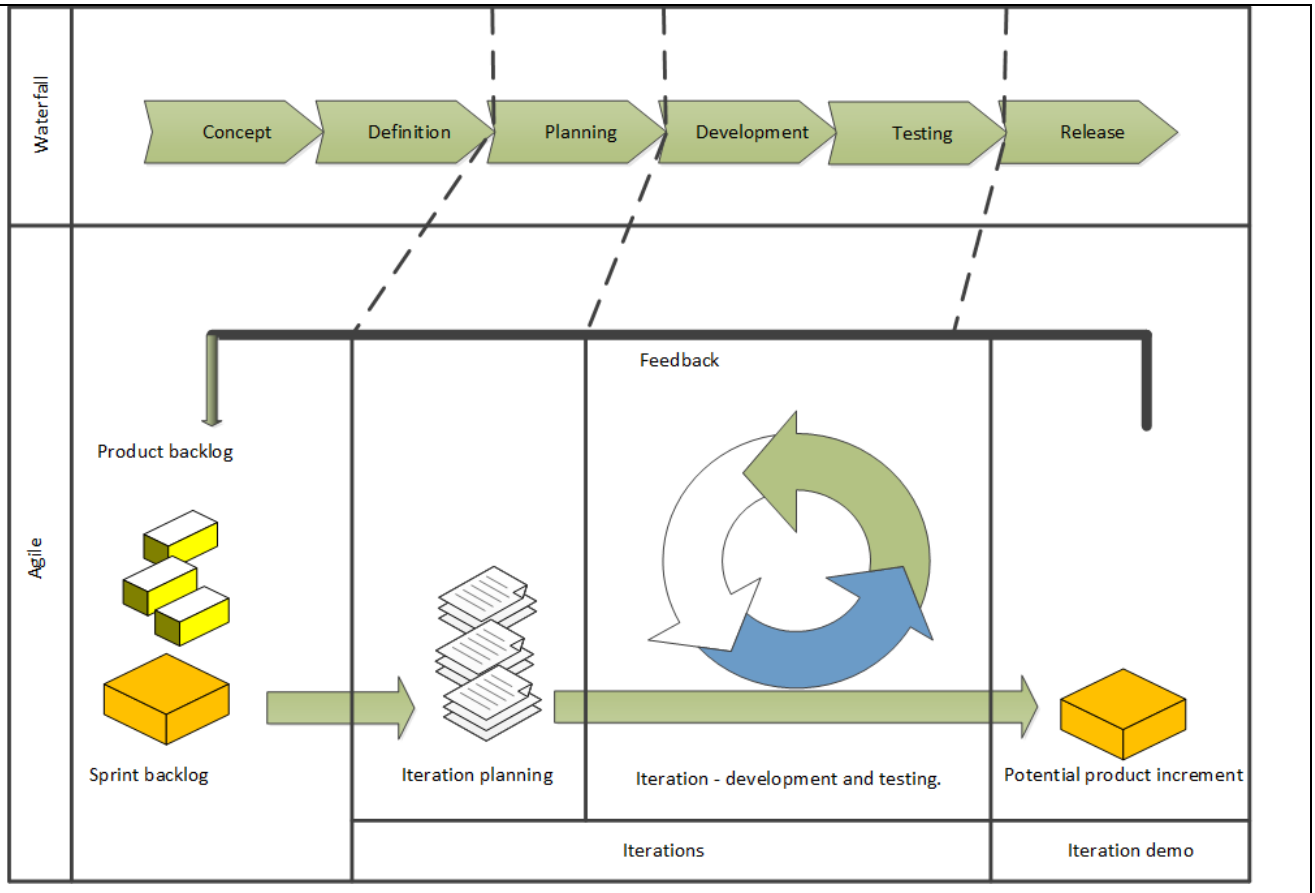


Figure 3. Scrum Activities/Concepts Mapped to Waterfall Software Development Methodology

The concept and definition phases, called the punch list, consisted of the product backlog. The product backlog consisted of high-level requirements in the form of user stories that could be customer-oriented or technical in nature. If a developer wanted to include a coding framework, it would be considered as a technical story. User stories could come from customers, developers, management, technical support, marketing, and customer service. Additionally, defects or changes to functionality introduced in previous iterations could be introduced as technical stories. Stories were assigned a business value and categorized as 1) "Must have—critical", 2) "Should have—it is a good thing to have"; or 3) "Delight—the wow factor". The firm viewed product backlog as a living document that included project plans but lacked specific dates. Requirement changes were added to the product backlog. The product owner prioritized the product backlog so that the team could focus on the most valuable features.

Release or sprint planning, a forward-thinking activity, captured what needed to occur in the application's next iterations. Specific stories were assigned from the product backlog to the sprint backlog. Infrastructure tasks needed

for future development were allocated to the sprint backlog. User stories were evaluated to identify any coding or other dependencies that may exist.

As part of sprint planning, developers were solicited for their input on the number of points each story should be assigned. Points signified the amount of time required to complete a story. Initially, a baseline estimate was used for a given story by the Agile team. In the early stage, the Agile team discussed the stories using points only. The estimate of the number of hours required to complete a story were not allocated by the Agile team. Spreadsheets were used by the Scrum master to create the release plans. Use cases were developed by the Agile team at the end of planning and provided enough definition for developers to proceed.

Table 2: Participants for Each Activity in the Scrum Process

		ISF participants					BF participants			
Activity	**Description**	**Scrum master**	**Developers**	**Testers**	**ISF manager(s)**	**ISF business analysts**	**Marketing lead**	**Marketing (whole team)**	**BF managers**	**Customer**
Product backlog	Prioritize a collection of user stories.	X				X	X			X
	Assign IT estimation points.	X	X	X		X				
Release planning	Estimate the number of iterations that can be performed before the release date.	X				X	X			
Iteration planning	Provide more details on the story and estimate the hours to complete the associated tasks.	X	X	X		X		X		
Daily standups	Daily standup meeting reviews the status of the iteration on a daily basis.	X	X	X	O	X	X	X	O	X
Daily Scrum	The set of activities that are done on a daily basis.	X	X	X		X		X		
Iteration demo	Agile team presents a demo of accomplishments during each iteration.	X	X	X	X	X	X	X	X	X
Iteration retrospect	Team reviews iteration and generates lessons learned to be applied to future iterations	X	X	X	X	X	X	X	X	X

Key: X indicates required attendance; O indicates optional.

During the iteration, many different development and testing activities occurred. Business process flows were documented, testing scripts were developed, software artifacts were developed, and other types of documentation created. Activities often occurred simultaneously since the Agile team members were co-located. The iteration demo, which occurred at the end of each iteration, was used by the Agile team to demonstrate their accomplishments. The demo showed developed code or demo interactive sections of webpages. The demo helped elicit feedback from the stakeholders. Changes suggested at the meeting were added to the product backlog. Additionally, iteration retrospectives were conducted to gather lessons learned from the previous iteration; the lessons were applied in future iterations.

An Agile mentor visited the Agile team every month to encourage them and to provide solutions to challenges and roadblocks. The entire team went through training together and learned concepts and skills. They became familiar with Scrum methodology and activities.

The Agile team mapped Agile processes to the Waterfall processes that the organization customarily followed. The definition phase was used in the release planning activity (see Figure 3). The list of items in the product backlog required 10 iterations to complete. These iterations were completed over the planning, development, and testing phases of the Waterfall methodology. The final deliverables for the project were integration with services developed by other teams using the Waterfall approach. Deliverables from the Agile team were integrated and tested holistically at the system level.

The Agile team conducted iteration planning on the first day of each three-week iteration. Initially, iteration planning typically took six hours to complete since the participants' expectations varied greatly. However, by the fifth or sixth iteration, the iteration planning activity was completed in less than three hours due to shared understanding and expectations. Through practice and discipline from the stakeholders, an understanding of the requirements evolved. Table 2 shows the Scrum process and lists the participants for each activity.

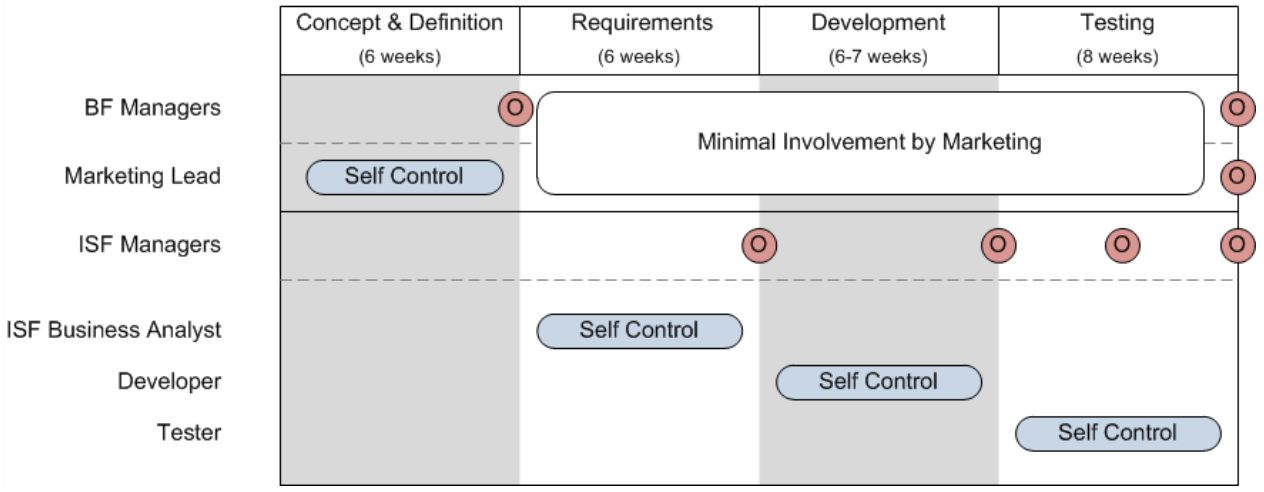
Daily standup meetings were an integral part of daily work during each iteration. The facilitator asked three questions at the daily standup meeting: what each team member accomplished the prior day, what would be done the current day, and what roadblocks, if any, needed to be addressed before proceeding.

The meeting served as an opportunity for those with expertise to help solve problems presented. If there was not a resolution in the Agile team, the Scrum master would take it beyond the team (e.g., problems with the server environment or its configuration). Initially, the daily Scrum took forty minutes to an hour, but, as the team gained experience, the length of the daily meeting was reduced to ten minutes.

IV. CONTROL IN SOFTWARE DEVELOPMENT

Control in Waterfall Method

Previous literature on control mechanisms in software development have primarily focused on sequential software development methodologies (Kirsch et al., 2002). Although a few studies have investigated specific control mechanisms for different software-development activities (Kirsch, 1997), they do not map control mechanisms to specific SDLC phases. Also, the interaction of various control mechanisms employed by the various contributors to the development process has received little or no attention in previous literature. Based on the data collected from the organization we examined, we present a framework that shows how the BF and ISF applied control mechanisms during the Waterfall software-development methodology (see Figure 4) prior to implementing the Agile methodology. In Figure 4, outcome controls (red circles with O) play an important role at the end of each stage and self-control plays a key control role in-stage.



Note: The Os represent points where outcome control was exercised

Figure 4. Control in Waterfall Software-Development Methodology

With a Waterfall approach, management typically applies outcome control at the end of each phase to understand the progress made in the project. Marketing leads, which are responsible for eliciting business requirements from the customer, rely on self-control. Self-control ensures these marketing leads satisfactorily complete their project responsibilities . Marketing managers apply outcome control at the end of the concept and definition phase, ensuring that business requirements are finalized and delivered to the ISF. Thereafter, the BF is only minimally involved until testing is complete. At this point, the BF becomes involved in "Go/No-Go" meetings that determine whether the project should be deployed to the customer. Lack of direct involvement in the requirements, development, and testing phases gives the BF little understanding of why the final product is the way it is. Based on our interviews, the BF (including marketing) did recognize that they lacked control during a substantial portion of the software-development process. This lack of control resulted in miscommunications and misunderstandings between the ISF and BF:

> *We wanted to have more say so and input to what was being defined versus what was delivered.* (Marketing Manager 1)

> *[With Waterfall software development], you get lot of finger pointing. [After development] the business sees what is going on [and] would say that's wrong, that's not what we want. And [then] IT would say…it's working as designed.* (Marketing Analyst 1)

In the Waterfall methodology, the ISF is principally responsible for activities during the requirements, development, and testing phases. Individual contributors, such as IT business leads, developers, and testers, receive what tasks they should perform. Self-control is the primary control mechanism that the individual contributors use. IT managers principally use outcome control mechanisms with their contributors during requirements, development and testing:

> *[Managers] don't know about the status [of development]. The requirements are done and we get the requirement and we have 6 or 7 weeks' worth of development time and during those 7 weeks we don't give any [status reports], they don't ask [for] any.* (Developer 1)

After the completion of the testing phase, IT managers also participate in "Go/No-Go" meetings. During deployment, the IT manager consistently monitors the configuration engineer's progress in deploying the software to a production environment.

Figure 5 summarizes the modes of control used in the Waterfall software development methodology. We observed that both IS and BF managers control the Waterfall methodology by controlling those who actually performed the task in their respective functions using outcome controls. However, individual ISF contributors applied self-control in contrast to outcome control applied by individual contributors on the BF side. The BF's lack of involvement during the software-development process's core periods and reliance on outcome control mechanisms leads us to conclude that the ISF largely controlled the Waterfall software-development process.

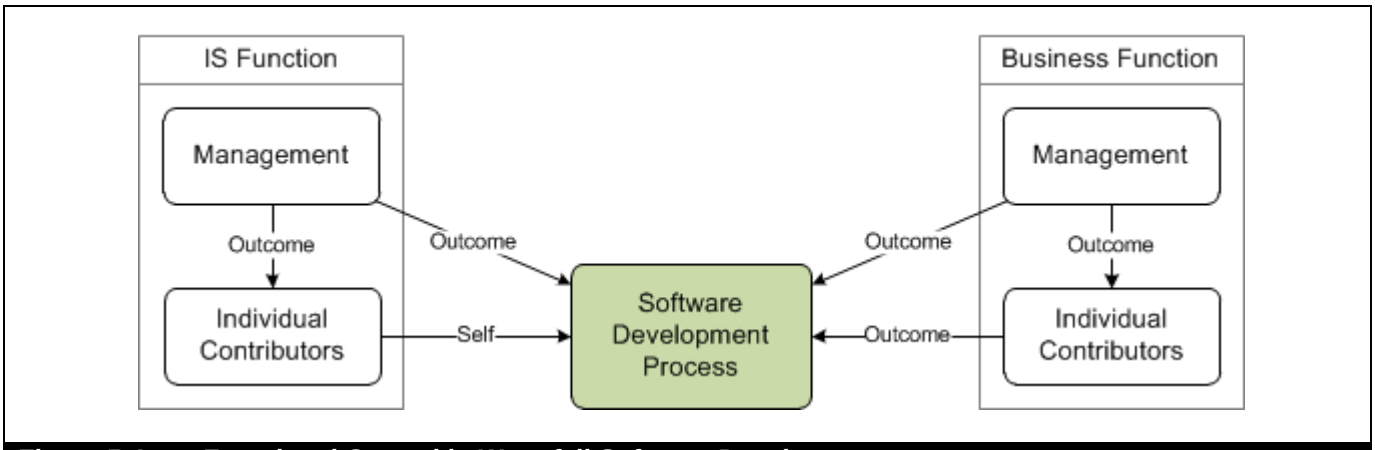


Figure 5. Inter-Functional Control in Waterfall Software Development

Control in Agile Development

To better understand and characterize control mechanisms used in Agile software development, especially by different controller and controlee roles, we analyzed our case data to derive a description of control in the Agile context. Table 3 summarizes the activities, controller and controlee participants, and the control mechanism based

on our analysis. We found that Agile activities and relationships were woven together in a tapestry of controls where activities, relationships, and control modes were interlaced and overlapped with each other.

Product Backlog—Prioritization

Two primary activities in Scrum are related to the product backlog: prioritizing a collection of user stories, and assigning an estimate of effort to complete those particular stories. Contrasted with the Waterfall software-development method, the product backlog allows the marketing lead to dynamically specify the list of stories that should be worked on for any given iteration. While much of the task can be done alone, the marketing lead is dependent on the Scrum master to provide additional insights into stories that may be coupled for efficiency reasons. The stories may also suggest technological feasibility (i.e., infrastructure readiness, dependencies on other systems) of specific stories. The Scrum master at the organization we studied characterized interaction between himself and the marketing lead as follows:

> *Marketing [was] responsible for the prioritization, but what the team works on was a decision that the team made as a whole. This is because there could be certain stories that share a common theme and/or a code base and so it becomes more logical to bunch them together possibly in the same iteration.* (Scrum master)

During the prioritization of user stories, at which time additional information is needed from the Scrum team, the marketing lead exhibits outcome control because emphasis is placed only on the outputs of the activity rather than being involved throughout the activity.

Product Backlog—Estimation

Estimation, the second task involving the product backlog, involves interaction between the Scrum master, software developers, and testers. The Scrum master seeks clarification and fine-tunes the time estimates provided by the developers and testers. Essentially, the Scrum master, who controls the process, is engaged with estimation. The Scrum master provides input and assists the team as necessary. The Scrum master at the organization we studied described the interaction:

> *The key concept is that the whole team is present when the effort is estimated. But there could be instances where an assumption on a part of work is causing overestimation or underestimation. It is the responsibility of the Scrum master to ensure the estimates are correct by having the marketing leads or members provide clarity around story definition. The Scrum master is responsible to collect all these details and store them in the Product backlog.* (Scrum master)

Table 3: Control Mechanisms involved for Scrum Activities			
Activity/mechanism	**Primary controller**	**Primary controlee**	**Dominant control mode**
	How mechanism is used to regulate behavior		
Product backlog—prioritization	Marketing lead	Scrum master / development team	Outcome
	Marketing lead dependent on Scrum master and team to provide information related to stories, and relies on that information without understanding or being involved in monitoring that process.		
Product backlog—estimation	Scrum master	Developers	Behavior
	Scrum master understands developers' concerns and needs with regards to estimation.		
Release planning	Marketing lead	Scrum master	Outcome
	Marketing lead is dependent on Scrum master for details related to how many iterations are required to complete project.		
Iteration planning	Scrum master	Developers	Behavior & hybrid
	Scrum master maintains knowledge of the planning process, is involved with it, and understands what must be done to further refine the user stories; Scrum master understands developer concerns and provides solutions that developer can effectively use.		
Daily standups	Agile team	Agile team	Clan
	Team members hold each other accountable for the progress of development activities; shared values and communication standards/vernacular emerge over time; acceptable behaviors are reinforced; team members understand each other's personalities, mannerisms, and behavior (including nonverbal behavior).		
Daily Scrum	Agile team	Agile team	Clan & self
	Team members perform development tasks individually or in conjunction with others; many adopt clan values into personal work habits; most individuals are motivated based on daily reporting to self-monitor in anticipation of future, frequent reporting.		
Iteration demo	ISF and BF managers	Agile team	Hybrid
	ISF and BF managers view the software produced during an iteration to see what stories were completed and the progress through a demonstration.		
Iteration retrospect	ISF and BF managers	Agile team	Hybrid
	Entire group generates lessons learned to be applied to future iterations; managers are primarily concerned that the team generates lessons learned and make plans for improvement but generally are allowed minimal input on specific changes that should be made.		

The Scrum master does not simply accept the estimates provided. Rather, this individual uses their experience of the process to advise and assist the team (Henderson & Lee, 1992). Engagement with the developers and testers provides the Scrum master the ability to monitor and evaluate the behaviors of individuals on the team. The Scrum master's active and knowledgeable participation is evidence of the fact that the Scrum master applies behavioral control during the estimation process.

Release Planning

During release planning activity, the number of iterations required to complete the stories in the backlog is decided. The marketing lead consults with the Scrum master and justifies the estimates provided by the team and the stories that may require additional time to complete.

At the organization we studied, the Scrum master's responsibilities in justifying the stories for a particular release were as follows:

> *It is up to release planning in conjunction with the Scrum Lead to ensure that teams are at least addressing the stories on the release plan and justifying stories that cannot be completed or need further granularity of work.* (Scrum master)

The marketing lead is dependent on the Scrum master to raise issues about the ability to complete stories in a given release even though a deep understanding of the complexities or how the team arrived at that conclusion may not exist:

> *After all story and task decisions are made and if a story is deemed impossible to complete in a release, the Scrum lead communicates this to the release planners.* (Scrum master)

The marketing lead is dependent on the Scrum master to provide the estimate of stories and justification for those stories. The marketing team's emphasis is on the team's outputs (Maruping et al., 2009a) without explicitly understanding or monitoring the team's behavior. Thus, outcome control is the dominant control mechanism in the release planning activity. The dependency of the marketing lead on the Scrum master and team and their interactions were described thusly:

> *The greater amount of control over the team's work plan is held by the release planning team with teams only having the ability to influence the plan through advanced planning. Teams give high level estimates on or assign points to the stories during their regularly scheduled planning activities. Release planners take those estimates, coordinate stories included in a release, and publish release plans. Teams work to the published release plan [with some modifications based on complexity discovered during iteration planning] and communicate with the release planners through burn down charts and demos on their progress.* (Scrum master)

Iteration Planning

The iteration planning activity requires close coordination between the Scrum master and the developers to negotiate the number of hours required to complete a particular story. The Scrum master overviews the iteration planning activity and the coordination that exists between the Scrum master and developers:

> *"Teams plan the next iteration at the beginning of the current or current -1 iteration. Planning activities are held at regular intervals to ensure that work is being adjusted based on need and scheduled consistently. The goal is that work is flexible until the beginning of the iteration; after that, it is theoretically fixed. Teams decide or evaluate the number of points they can perform in a sprint of a fixed time length. This number is adjusted based on a team's performance over time. Teams take the release plan that is based on their team's assignment of points to a story in the previous planning sessions and take a deeper look at the stories with business owner input. From this work, they determine what tasks must fall out of the iteration and should be pushed to the next or if the task is worked."* (Scrum master)

The prioritized list provides a list of stories that are to be completed during the iteration. Once the list of stories is solidified and broken into tasks, the Agile team cannot change the direction of the work for that iteration:

> *Before an iteration begins the team as a whole decides what stories to work [on] and breaks them into tasks. Once this decision is done, then this is locked and loaded—no change in direction until the end of the iteration.* (Scrum master)

The Scrum master is actively involved in the iteration planning process and interjects at appropriate times to solicit additional details or to ensure that stories are appropriately sized. The stories are assigned to an iteration and the work is completed. A developer explained the involvement and monitoring of the iteration planning:

> *The Scrum master was involved, but the developer did most of the estimation. The Scrum master did pitch in when he felt that the story was too big and needed to be broken down into smaller stories.* (Developer 2)

The Scrum master explained how the process is monitored and when the Scrum master may need to intervene during iteration planning:

> *Primarily the person that takes on the particular task is the one that is estimating and if there are stories that are broken into way too many tasks that is causing overflow of resource in a iteration, the Scrum master ensures that the story causing this overflow gets simplified or moved to an another dedicated iteration. Ultimately all the data—stories targeted for an iteration, tasks that make each stories, resource assignments, time estimates and other details get recorded in to the Sprint backlog to create the burndown chart.* (Scrum master)

The Scrum master, as a controller, is cognizant of the developers' behavior and performance; this individual monitors and evaluates their behavior in order to assist them. The Scrum master exhibits behavioral control over the developers during iteration planning.

We also observed that ISF and BF managers saw a need to lock down concrete requirements for each iteration. They identified the need for a priori, documented, and well-defined requirements that could not be changed during the course of a particular iteration. This request appears to be a fallback from the purely fluid change of requirements in iterations typically cited as present in Agile projects under emergent control. Thus, we see the presence of a more planned and rigid approach to dealing with requirements. Iteration planning is the first observed indication that Agile, as implemented in an organization transitioning from Waterfall, uses a form of hybrid control.

Daily Standups

Daily stand-ups require bonhomie among an Agile team's participants. The meetings generally result in team participants answering questions related to what they accomplished the previous day, what was planned to be accomplished the current day, and what roadblocks might exist. Initially, the idea of being accountable to peers was met with trepidation. However, in a short time, team members started acknowledging the benefits of holding each other accountable and having shared awareness of the team's activities:

> *Stand ups are a blessing and a curse to an Agile team. They provide daily input into the teamwork and bring up roadblocks or issues in a timely manner, but it is also seen by some as micromanagement, which creates conflict in some groups. Over time, teams get into a cadence of these meetings and they become either more productive and collaboration increases or fall apart. This is usually based on the commitment of management to ensure team participation.* (Scrum master)

> *Initial inhibitions were removed as the team matured and worked with the other team members. Team members after the first 2 iterations were offering help, suggestions and helping out in removing any complexities. Agile training for the whole team as a whole worked really well, in that it brought along a common sense of understanding and the use of common terminology. This paradigm shift…helped a lot.* (Scrum master)

Shared awareness extended beyond just the development team. Marketing was also involved in the daily standups and, compared to previous projects, understood to a greater degree the current progress on the project and the challenges that the team encountered:

> *Marketing always knew what we were working, since they were collocated, and moreover the daily standup / Scrum detailed…progress made.* (Scrum master)

Daily standups provided team members an opportunity to hold each other accountable for the work that was to be completed and also provided a regular interaction where shared vocabulary, expectations, and clan culture emerged:

> *Team members are more apt to offer help to other members in a standup than if left to their own. Standups facilitate communication in a team and by getting to know other team members with the daily contact, communicated styles are recognized and adapted to. Peer pressure seems to be the mechanism of accountability for a group. No one seems to want to voice failure to other members of the group.* (Scrum master)

The culture that emerged from the daily standups was relaxed and team members felt comfortable sharing progress and receiving feedback. However, team members still held each other accountable for assigned tasks. Over time, there was a decrease in illegitimate excuses offered when work was not completed:

> *Standup meetings are informal and all the team feels comfortable in attending and sharing things.* (Developer 1)

> *Team members were very comfortable asking questions about and providing constructive criticism of the work completed.* (Developer 2)

> *[There were] a few instances where during the daily standup, someone would use excuses as to why some work was not completed, [but] over the course of a few cycles, the excuses were no longer made, and work was completed as planned. Or, in cases where it wasn't, the excuses were legitimate.* (Core team lead)

As a shared culture emerged, team members established a team identity. Team members began focusing on helping the team achieve its goals. At the same time, members remained responsible for their own goals:

> *The entire team was pretty dedicated. In case there was any issue with task completion, other team members would pitch in to clear the obstacles/provide help as needed.* (Developer 2)

The daily standup created an environment where participants were monitored by their peers and helped to reinforce behaviors consistent with that vision (Kohli & Kettinger, 2004; Ouchi, 1979). The Agile team's members encouraged each other by offering suggestions on how to fix issues. They also acknowledged each other's contribution to the project. The team members used acronyms and vernacular that gained acceptance with the team over time. Thus, clan control was the dominant control mechanism exhibited during daily activity.

Daily Scrum

The clan culture pervaded the team's daily work. Since the Agile team was co-located, issues and concerns that arose during the daily Scrum were discussed in an informal manner. Peers were able to help each other out without specifying any formal control mechanism. The group took care of each other and was concerned with accomplishing their work, and with ensuring that others in the group completed their work:

> *I saw more open dialogue between team members that was more focused on getting the work done that they were responsible for on a daily basis rather than on making sure they "covered themselves" in case work didn't get done. An analogy that I like to use to represent this attitude within our Agile team is that of a sports team, that once ahead, continues to play the offensive game that put them ahead instead of becoming defensive-minded hoping to "hold on to a win" and ultimately loses the game.* (Core team lead)

Co-location decreased the cost of communication among team members and enabled access to information more rapidly than under the Waterfall method. In the case of information exchange involving the marketing function, the marketing lead's engagement was stronger. There was a greater influence on the process than under the previous SDLC development approach:

> *Co-location translates to the marketing lead engaged from day 1 of the project so purely by nature they are engaged on a day to day basis. Because of this any changes to the targeted stories were evident with necessary correction applied as and when needed.* (Scrum master)

Clan control is the predominant control mechanism the Agile team used in their daily Scrum. The ensuing culture, marked transparency, and visibility fostered more peer interaction and extended beyond holding each other accountable. It held the team to higher expectations that required team members to work together:

> *Agile teamwork seemed to foster more peer programming activities along with constant conversation and feedback. Team members seemed more likely to talk with others on issues and solicit help than in a non-Agile group. The transparency and visibility creates an atmosphere of community which helps to make asking for help more acceptable.* (Scrum master)

Iteration Demo

The iteration demo provided a venue for IS and business managers to review the work done by their contributors. The agreed-on stories developed and tested by the Agile team resulted in a prototype scrutinized by management. Managers took this opportunity to evaluate contributions of the Agile team members:

> *The iteration demo is restricted to purely look at what the team has accomplished and to take away the progress from a management perspective. Feedback was welcome. If there were additional stories, those went to the product backlog to get prioritized appropriately. Team members were autonomous and were shielded from being pulled in different directions. The Scrum master combined with the business partners were responsible to ensure that the team was progressing in the intended direction. Showcasing is a venue to show off work and not a forum to question priority. Managers would have needed to contact the Scrum master to know what was [being worked on].* (Scrum master)

In some activities, managers simply viewed the outcome of the iteration and were not involved in the daily activities of the team:

> *Management had little involvement with the day-to-day activities. They were mostly interested in the broader picture of the entire iteration and the demo at the completion of the sprint. Discussions were held with management on the workload.* (Scrum master)

During the iteration demo, managers did not question the stories that were completed or how long it took to complete a specific piece of functionality. Rather, they focused on what was accomplished during the iteration:

> *Managers never questioned the story prioritizations. In every retrospective, we would announce what we were going to work on next, so they know what we were going to work next. There was not a whole lot of involvement from the management in the day-to-day activities during the iteration. I did discuss my current work load [not an Agile related responsibility] with the management, but not a lot about the Agile project.* (Developer 2)

> *In my experience, no manager has questioned the priority of a story or the number of hours to complete the story in the retrospective/demo. The questions that were asked in the demos usually related to the accomplished work and how it could be improved.* (Scrum master)

Management was also interested in comparing the experiences and results of using the Agile methodology with the Waterfall approach:

> *As expected, [managers] asked questions about the functionality that was being built, and sometimes wanted to see things implemented differently in some cases, which we did. But, a large number of the questions revolved around the efficiency, progress and effectiveness of the Agile environment—since using Agile on this project was a pilot test of the development methodology at our company. Management was continually interested in knowing how the progress of the Agile team stacked up to the progress of other Waterfall projects that the company had undertaken in the past.* (Core team lead)

The iteration demo exposes another facet of hybrid control (i.e., the presence of feedback at regular intervals). During Waterfall development, the outcome control exists and evaluation of the software is often done only once and against a priori documented requirements. When Agile was introduced in this organization wedded in a Waterfall tradition, managers provided feedback at regular iteration intervals. In a purely emergent control approach, team members and managers take corrective action to the software on a continuous basis (refer to Table 1). In hybrid control, however, only managers provide feedback at iteration demo. The feedback is documented and incorporated in a later action during the next iteration planning activity.

Iteration Retrospect

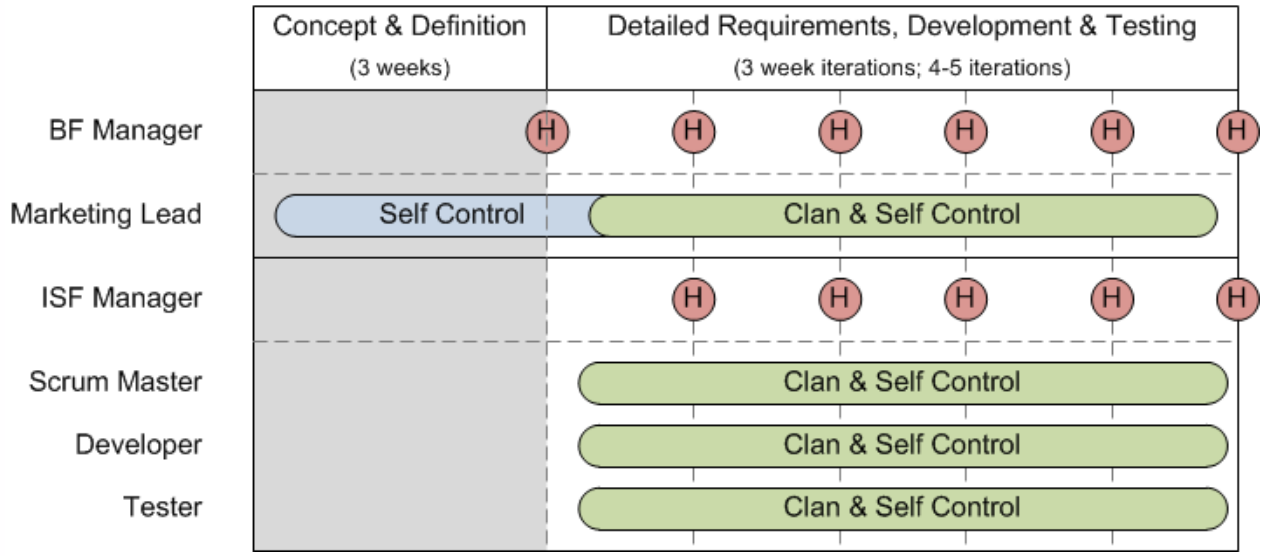
The lessons learned during the iteration were discussed at the iteration retrospect attended by the Agile team and management:

> *During the iteration retrospectives, management asked both general and detailed questions of the team. They were active participants in that they constructively questioned the work of past iterations, and freely offered their recommendations on what to change and what new functionality they would like to see in the next iterations. Management was agreeable to listening to the recommendations put forth by the Agile team for the next iteration, but almost without fail, they requested reprioritization of stories, changes to functionality, or new features that we'd not planned on implementing.* (Core team lead)

During the iteration retrospect, the discussion centered on the lessons learnt about preset requirements. Management was minimally involved in the Scrum process and engaged the Agile team around iteration's outcomes. Management also established lessons and planned future improvements. Hybrid control formed the predominant control mechanism since activity was motivated around outcome-oriented discussion.

In summary, the Agile team used all activities prescribed for the Agile software-development method. However, unlike a purely emergent control context, we also observed their need for a priori documented requirements during the iteration-planning phase. During the iteration, the Agile team could not make any changes to the requirements and was required to follow the prescription in the documentation. BF and ISF managers provided feedback only during the iteration demo and iteration retrospect as opposed to emergent control's continuous feedback during and after each iteration. These differences are reflective of hybrid control, and they represent a confluence of the properties of outcome control used in Waterfall software development and emergent control, which might explain pure Agile software development outside of a Waterfall transitional situation.

Figure 6 summarizes the location of the control modes as discussed above.



Note: The Hs represent points where hybrid control is exercised

Figure 6. Control in Agile Software-Development Methodology

Changes in Interfunctional Control Responsibilities and Influence

From an inter-functional control perspective, we observed that developers, testers, and business analysts from ISF and BF were closely involved throughout the iteration process. They participated in daily standups, iteration planning meetings, and meetings at the end of the iteration. In the Agile paradigm, marketing analysts are members of the Agile team and, thus, interact with the developers and testers as active participants in the entire Scrum process. The Agile team members developed a bond that helped to foster a spirit of teamwork, which inspired them to work closely toward the project's success:

> *Marketing involvement is high. They were involved in our daily meetings. Marketing was not like, "here are the requirements, take this and go". They were involved in our daily [activities]. They gave us continuous feedback on [those activities].* (Developer 1)

> *Agile is so fast, so productive and so accountable and it is fun too. You don't get to meet with Marketing folks all the time. You also get rewarded pretty well because if you do good things for them they will say you did a good job. But those things don't happen in the [Waterfall] process.* (Developer 1)

In the early stages, daily stand-ups took 45 minutes to an hour to complete. As iterations progressed, they only took 10-15 minutes. By then, there was complete understanding among team members of what was expected of them and they were able to express their progress in a way that the Agile team's members understood well. The team members felt the need to perform at top capacity. Team members also supported each other in their work:

> *In the Waterfall process for some days if you don't want to work, if you don't feel like working, you can take off and do some other things. But [with Agile], the next morning you have to report what you did the previous day. So you have to work.* (Developer 1)

A clan mentality soon developed where each member understood their functions and the functions of other team members. The marketing leads in the Agile team experienced a better understanding of the situations that the developers went through to get the software built. They gained a first-hand understanding of the need for proper environment setup, test cases, and issues related to coding, versioning, and scoping:

> *In the daily status meetings, we discussed the problems we are facing. They never heard of those problems during the [Waterfall] process. Because once they give their requirements, they don't come in contact with development at all. [With Agile], they know what kind of problems [exist] and if we put more time on a*

particular story, they understand why it is taking more time. Before they [would] say, to do this task, why do you need so much time? But now they are aware of what is involved in the development process. (Developer 1)

Such close involvement allowed for better scoping in terms of what stories could be added per iteration:

So [marketing] knows that problems that development raises are changing these. Some things cannot be changed so drastically. So they will make changes to the requirements. So they will think about how development will react and are more sensitive. (Developer 1)

Developers envisioned the end product characteristics that marketing desired. The iterations helped them dynamically pace their work to attain the goal:

In the [Waterfall methodology] when requirements are finalized, there will be a freeze date after [which] no more requirement changes [are allowed]. But [with Agile], it is not like that. It is continuous and requirements can be changed. So, Marketing can change their requirements anytime. So, they can fine-tune the requirements. Once we build this, 'oh this is not what I thought', so let's change it. In the [Waterfall] process, once the freeze is in, [marketing] is out of [the process]. They had to wait for the final product [to] come in. (Developer 1)

The ability to exercise hybrid control through feedback at regular intervals and documented requirements helped BF better understand the software-development process. Further, the lock-down of the iteration provided ample time for the BF to gain a good understanding into the issues faced by ISF. The ability to tweak the stories from one iteration to the other, with close interaction and cooperation of the developers and testers, helped the Agile team to deliver a quality product that closely matched their customers' expectations.

...[Agile] reduced the defects that actually showed up in production. [It also allowed us] to have more say so and input to what was being defined versus what was delivered. (Marketing analyst 1)

Individual marketing analysts relied on clan and self-control to ensure satisfactory completion of their responsibilities toward the project. The Agile methodology provided the BF with the ability to leverage their requirements and the ability to control the software-development process to achieve a product that was closer to customer expectations than ever before:

The more you go into the process of Agile, the more you get into it: communication, frequent inspections, adaptations, excellent team work, self-organization, accountability. You get high-quality software. (IS Manager 1)

Figure 7 summarizes the numerous modes of control used in the context of an Agile software-development methodology[2]. An IS manager still retained control of the allocation of their resources (e.g., deciding which resource would work on a given team) to specific development projects; however, they may lose control (or may need to loosen control) over some of the daily activities of the software-development team. For example, marketing can shift the daily activities of an IS manager's resources by assigning them to user stories without any intervention by the IS manager. Note that the overall control responsibilities are now viewed as more balanced between ISF and BF with the software-development process moving closer to the BF.

[2] Note that control can be conceptualized both as mechanisms exhibited between actors who perform different functions and as control that any single actor has on the software-development process itself.

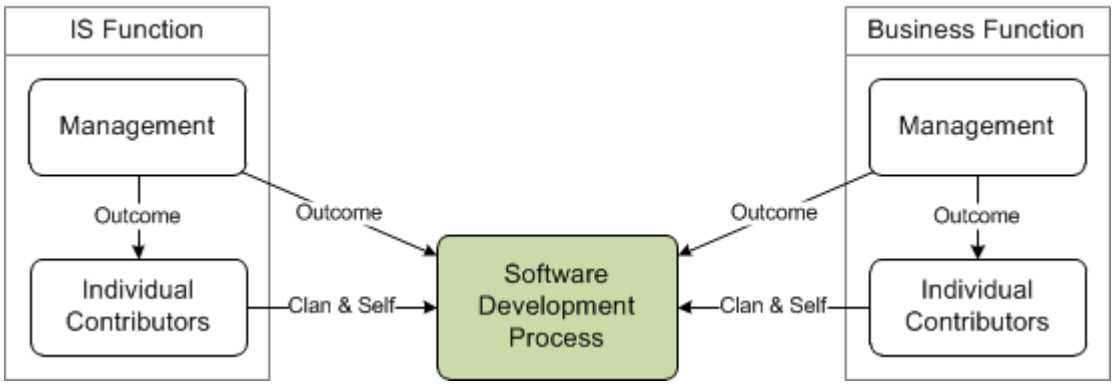


Figure 7. Inter-Functional Control in Agile Software Development

V. PROPOSITIONS

As we discuss in Section 3 and depict in Figure 8, introducing Agile methodology facilitates the breakdown of requirements into stories, which are then prioritized and assigned for completion to one of the iteration cycles. In a hybrid-control context like the one studied here, the requirements are locked and loaded for each iteration. The evaluation of the iteration at regular intervals provides an opportunity for the stakeholders to understand the requirements and for supervisors to provide feedback (see Figure 8). The feedback received from supervisors is documented as requirements and incorporated for implementation for a later iteration cycle.

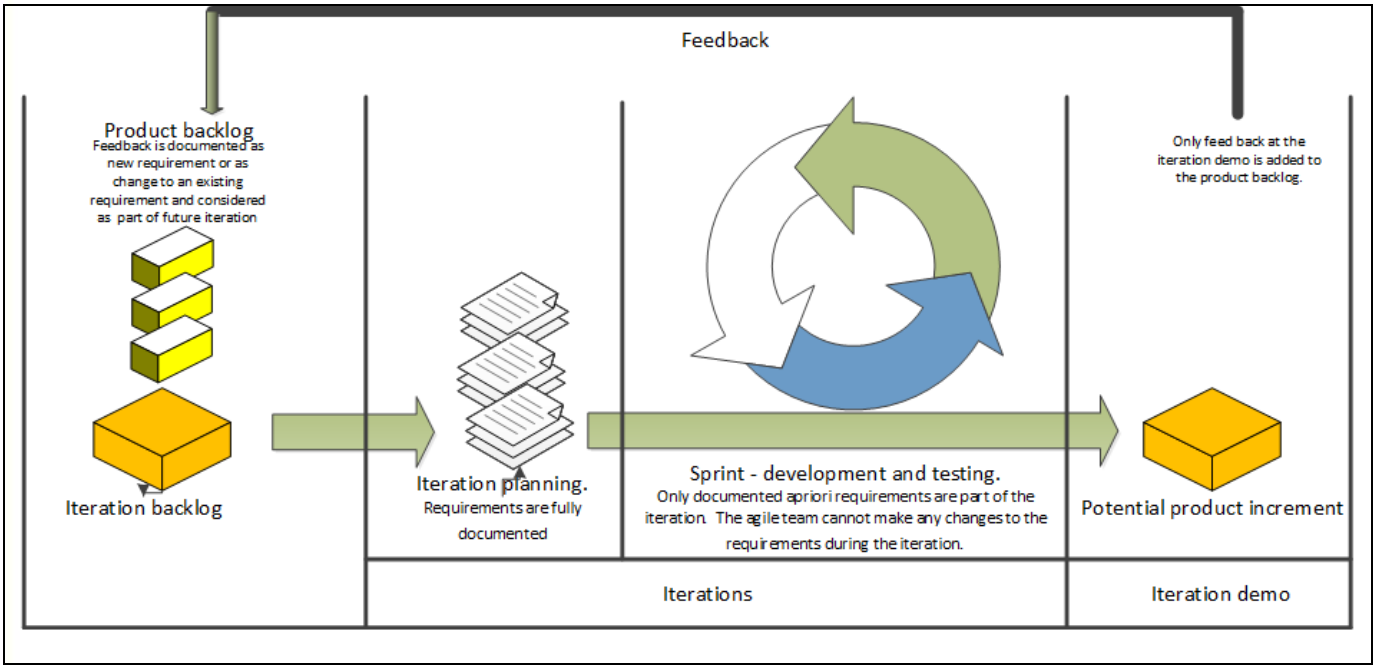


Figure 8. Scrum Activities/Concepts as Recognized by Hybrid Control

A different control context is established by the presence of pre-set and locked specifications for each iteration and feedback only at iteration intervals rather than continuously. The control context is different when compared to outcome control during the Waterfall methodology, and emergent control as described in a purely Agile scenario. Table 4 extends the comparison of the control mechanisms presented by Harris et al. (2009) to include hybrid control.

Table 4. Comparison of Outcome, Emergent, and Hybrid Control					
Control	**Purpose**	**Frequency**	**Evaluator**	**Construction of standard**	**Comparison**
Outcome control	Evaluation	Once	Manager	A priori	Completed projected versus specification
Emergent control	Corrective action	Continuous	Multiple stakeholders	Evolving by stakeholder	Emergent outcomes versus tacit specifications
Hybrid control	Feedback	Regular intervals	Manager	A priori and supervisor feedback	Completed iteration versus iteration specifications

Our research suggests that, in organizations with deep-seated "traditional" Waterfall software-development routines, introducing Agile results in a higher use of the hybrid control mechanism.

The Waterfall software-development method typically offers limited opportunities for interaction between the BF and ISF project contributors. The first significant opportunity for BF to exercise control occurs at the end of the concept phase when BF contributors interact with ISF contributors to transfer business requirements into technical requirements. The lack of IT knowledge often prevents BF project contributors from exercising any control mechanism other than outcome control. The controller-controlee relationship usually ends at the concept phase. As a result, the BF "hopes" the ISF will deliver a product that matches the requirements. During the testing phase of the Waterfall method, the BF has another opportunity to interact with ISF. The outcome controls used will not benefit the BF since software development is almost complete at this stage. Except for minor changes, the BF is stuck with the software that is given until the next cycle of software development.

The use of clan and hybrid control by the BF team members through close collaboration with ISF members throughout the Agile methodology is very different from the Waterfall method:

> *Marketing has [more control of the software-development process] because they are involved in everything. Marketing has persuaded IT to embrace Agile development methodology as a possible option. And that's the important thing.* (IS manager 1)

As we can see, the Agile development environment allows the BF to more closely align itself with the ISF. When the BF and IS align, both functions learn first-hand about each other's issues and priorities, with individual ISF and BF contributors engaging in self- and clan control. Managers employment of hybrid control interaction was heightened:

> *We are all responsible for getting this done, all are accountable and we wanted to test that theory. That really, really works in this environment.* (Marketing analyst 1)

We observed that Agile's introduction provided the BF greater visibility into the software-production process. The BF's ability to closely engage with the ISF helped shape an environment where changes to the requirements could be quickly incorporated on an iteration-by-iteration basis. Agile activities such as daily standups and the iteration demo put contributors in a situation where they were under pressure to perform and not let their colleagues down. Given that the BF and ISF contributors were co-located and interacted with each other on a daily basis, contexts and terminologies were well understood. As a result, less background information needed be given in meetings in order to have meaningful discussions. The constant interaction provides less need for outcome control since the issues were well understood by both functions. The presence of hybrid control, with its need for documentation, requirement lock down during iteration, and feedback at regular intervals provided a mechanism for the BF to understand the requirements. Hence, there was better control of the software-development process.

VI. DISCUSSIONS AND IMPLICATIONS

Some organizations may be reluctant to adopt Agile in a situation where multiple software-development methodologies (including Waterfall) must co-exist. We have observed that the use of control theory as a lens to study Agile phenomena lay the foundation for better understanding of how Agile's introduction unfolds in an organization steeped in the Waterfall development lifecycle. Our case data supports the idea of two distinct types of control mechanisms.

Control Mechanisms Exerted by the Controller Manager and Controllee Developer/Analyst

Moving beyond the previous research's recognition that BF exert outcome and ISF exert self-control in the Waterfall software development, we observed that, with Agile's introduction, both ISF and BF contributors used clan, self- and hybrid control to complete their tasks. The recognition that hybrid control can represent an amended Agile methodology that includes core concepts of the Waterfall method such as the need for concrete a priori requirements and requirement documentation is a significant contribution of this study. Our research shows that hybrid control helps controllers and controlees make better sense of their relationships when Agile was introduced in the organization we studied, which was deeply situated in Waterfall software-development methods.

Control Mechanisms Exerted by the ISF and the BF on Sofware Development Process

Agile contains control mechanisms that let the BF be a more dominant partner in the software-development process and allows software to be created that meets business needs in a timely fashion. Through daily standups and co-location with ISF, the BF understood and contributed to the team. For example, daily standups and iteration demos required better self-control on participants' part to ensure the project's smooth progress since the team members were dependent on each other to complete their tasks. Moving to a flatter and more peer-based project structure allowed clan, self-, and hybrid control mechanisms to permeate the software-development process.

Timing and Control

The time required to complete planning, development, and testing in the Waterfall software-development lifecycle was substituted with sprint iterations. The sprint cycles helped the BF to periodically evaluate the product backlog and dynamically specify the list of stories that should be worked on for any given iteration. This resulted in a more hybrid method of control. Such an arrangement helped keep important aspects of the Waterfall development lifecycle intact. Existing standards and business processes were largely followed, which thus made the effort understandable to the process controllers in the organization.

From a control perspective, our investigation into the activities involved in the Scrum process showed controller-controlee relationships that had not been explained in other literature. The feedback provided by managers at the end of iteration demo or iteration planning indicated the presence of emergent control. However, the fact that the feedback was documented and made part of another sprint cycle, coupled with contributors' inability to change the requirements in a iteration, resembled outcome control similar to the one existing during the Waterfall methodology. The proposed hybrid control mechanism bridges the feedback aspect at the end of each iteration cycle along with the documented a priori requirements present at the beginning of each iteration cycle.

VII. LIMITATIONS AND FUTURE RESEARCH

We studied a large organization with multiple IT divisions and multiple business divisions. The research addressed control relationships between individuals reporting to different divisions in the IS and BFs in their efforts to control the Waterfall and Agile software-development methodologies. The study addressed a single-site case study and, although revelatory in nature, some of our observations may be ascribed to the idiosyncrasies of a single site. Future research could investigate the impact of ISF and BF on the software-development process in organizations of varying sizes and complexities. We suspect that BF's influence on the software-development process would be lower in smaller organizations; however, in order to confirm this theory, additional research is needed.

Furthermore, the experiences we captured were in context of Scrum. Scrum is one of many different Agile methodologies, but one that is arguably one of the most widespread. Other Agile methodologies such as Extreme Programming (XP) and Agile Unified Process (AUP) employ different activities and, though we assert similarities in our overall findings based on informal discussion with practitioners and analysis of our previous experience, rigorous evaluation of cited contexts is needed.

Another limitation of our study was that, in our analysis, we considered only the initial implementation of the Agile project and the associated use of hybrid control. We did not fully capture temporal aspects and the evolution of control across multiple projects. Future research could undertake snapshots of control at different stages of familiarity with the Agile methodology individually and organizationally over multiple projects. Further research could also focus on the inter-functional impact of Agile methodology on software-development evaluation metrics such as time to market, software quality, development efficiency, and project performance based on which function (ISF or BF) controls the software-development process.

Finally, the idea of hybrid control requires validation across many Agile projects in an organization. In addition, the impact of the activities initiated by other Agile methods on outcome hybrid control needs to be systematically evaluated. We were able to investigate only a few areas. Understanding control mechanisms will ultimately allow

organizations to better achieve their objectives and, in some cases, allow organizations that were once hesitant to convert operations to yield the numerous benefits presented by Agile methodologies.

VIII. CONCLUSION

Many challenges exist when integrating Agile in an environment where there are highly structured developmental processes. From this revelatory case study, we observed how the implementation of a hybrid Agile methodology might benefit large organizations seeking to embrace Agile software development. Our research shows that control shifts from the ISF towards BF because the use of Agile methodology provides the BF greater visibility into the working of ISF. This helped the BF to better recognize the ISF's challenges and dynamically adjust requirements to remain congruent with project goals and capabilities. In doing so, the expectation gaps were better harmonized. Consequently, with Agile's introduction, the BF had more control over the software-development process when compared to the earlier Waterfall methodology.

In terms of introducing Agile, we describe a path by which organizations embedded in the Waterfall methodology can move to Agile development. Sprint iterations replaced the time allocated for planning, developing, and testing in the Waterfall software-development process. This change kept iterations understandable to the process controllers in the organization. Our research also identified touch points for managers to exert control over their contributors. The iteration demo activity provided an excellent opportunity for managers to monitor their team members relative to other contributors in the Agile team.

Our research highlights the presence of a hybrid control mechanism. The hybrid mechanism is unlike traditional outcome control in the Waterfall method, where feedback is provided once based on a priori, documented requirements. It is also unlike emergent control, where feedback is continuously provided by the stakeholders and documentation is not needed. In hybrid control, managers use the iteration demo and iteration retrospect activities of the Agile methodology to provide feedback. Feedback is documented and incorporated in future iterations.

Finally, although ISF management may have less control over requirement prioritization and the ability to determine which requirements are completed, the iteration demo activity provided ISF management capabilities to better monitor and reward their contributors relative to other members of the Scrum team. From an inter-functional perspective, Agile introduced new dynamics to the BF and ISF relationship. If harnessed properly, this dynamic could provide avenues for better software development.

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 paper 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., Conboy, K., & Wang, X. (2009). "Lots done, more to do": The current state of Agile systems development research. *European Journal of Information Systems*, *18*(4), 281-284.

Barlow, J., Giboney, J., Keith, M., Wilson, D., Schuetzler, R., Lowry, P., & Vance, A. (2011). Overview and guidance on Agile development in large organizations. *Communications of the Association for Information Systems, 29*(2), 25-44.

Boehm, B., & Turner, R. (2005). Management challenges to implementing Agile processes in traditional development organizations. *IEEE Software, 22*(5), 30–39.

Boehm, B. W., & Turner, R. (2003). *Balancing agility and discipline: A guide for the perplexed*. Boston, MA: Pearson.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer, 21*(5), 61–72.

Bose, I. (2008). Lessons learned from distributed Agile software projects: A case-based analysis. *Communications of the Association for Information Systems, 23*(24), 619-632.

Cao, L., Mohan, K., Xu, P., & Ramesh, B. (2009). A framework for adapting Agile development methodologies. *European Journal of Information Systems, 18*(4), 332–343.

Cockburn, A. & Highsmith, J. (2001). Agile software development: The people factor. *Computer*, 34(11), 131-133.

Dyba, T., & Dingsoyr, T. (2008). Empirical studies of Agile software development: A systematic review. *Information and Software Technology, 50*(9-10) 833–859.

Eisenhardt, K. M. (1985). Control: Organizational and economic approaches. *Management Science, 41*(2), 134–149.

Fitzgerald, B., Hartnett, G., & Conboy, K. (2006). Customising Agile methods to software practices at Intel Shannon. *European Journal of Information Systems, 15*(2), 200–213.

Haney, M. H. (2009). *Control of information systems development: Investigating the relationship between control and performance* (PhD Dissertation). University of Pittsburgh.

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

Henderson, J. C., & Lee, S. (1992). Managing I/S design teams: A control theories perspective. *Management Science,* 38(6), 757–777.

Jaworski, B. J. (1988). Toward a theory of marketing control: Environmental context, control types, and consequences. *The Journal of Marketing, 52*(3), 23–39.

Karlström, D., & Runeson, P. (2005). Combining Agile methods with stage-gate project management. *IEEE Software, 19*(6), 43–49.

Kirsch, L.J. (1996) "The Management of Complex Tasks in Organizations: Controlling the Systems Development Process." *Organization Science* 7(1) pp. 1–21.

Kirsch, L. J., Sambamurthy, V., Ko, D. G., & Purvis, R. L. (2002). Controlling information systems development projects: The view from the client. *Management Science, 48*(4), 484-498.

Kirsch, L. S. (1997). Portfolios of control modes and IS project management. *Information Systems Research, 8*(3), 215-239.

Kohli, R., & Kettinger, W. J. (2004). Informating the clan: Controlling physicians' costs and outcomes. *MIS Quarterly, 28*(3), 363–394.

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

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

McAvoy, J., & Butler, T. (2007). The impact of the Abilene paradox on double-loop learning in an Agile team. *Information and Software Technology, 49*(6), 552–563.

McHugh, O., Conboy, K., & Lang, M. (2011). Using Agile practices to build trust in an Agile team: A case study. *Information Systems Development,* 8, 503–516.

McHugh, O., Kieran, C., & Michael, L. (2008). *A study of the use and effectiveness of controls in Agile information systems development projects.* Proceedings of the 3rd International Research Workshop on Information Technology Project Management.

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

Ouchi, W. G. (1979). A conceptual framework for the design of organizational control mechanisms. *Management Science, 25*(9), 833-847.

Orlikowski, W. J. (1991). Integrated information environment or matrix of control? The contradictory implications of information technology. *Accounting, Management and Information Technologies*, *1*(1), 9-42.

Vinekar, V., Slinkman, C. W., & Nerur, S. (2006). Can Agile and traditional systems development approaches coexist? An ambidextrous view. *Information Systems Management, 23*(3), 31–42.

Yin, R. K. (2009) *Case study research: Design and methods (*Vol. 5). Thousand Oaks, CA: Sage.

APPENDIX A: STAKEHOLDERS IN WATERFALL SOFTWARE DEVELOPMENT

Table A-1: Stakeholders in Waterfall Software-Development Activities

Activity	Description	Stakeholders
Concept	Assess customer demand for the proposed concept and how it fits with the organization's strategy, technical feasibility, and profitability.	Strategic marketing, product marketing.
Definition	Guide cross-functional core team tasks and activities during the definition phase. The step results in a comprehensive business justification document (BJD), an "investor-quality" description of the proposed product/service and a plan to deliver.	Strategic marketing, product marketing, financial analyst, legal, audit, business analyst.
Planning	Provide more details on the story and estimate the hours to complete the associated tasks.	Product marketing, business analysts, requirements writer, product—SME, developer, tester, marketing (whole team).
Development	Meetings review the status of the iteration on a daily basis.	IT lead, tech lead, developer, business analyst, tester, marketing, managers, users (whole team).
Launch	The set of activities that are done on a daily basis.	Developer, tester, analyst, marketing .

APPENDIX B: STAKEHOLDERS IN SCRUM (AN AGILE SOFTWARE-DEVELOPMENT METHODOLOGY)

Table B-1: Stakeholders in Scrum Activities

Activity	Description	Stakeholders
Product backlog	Prioritize a collection of user stories.	Customer, Scrum master, business analyst, marketing lead.
	Assign IT estimation points.	Developers, Scrum master, business Analyst, tester, marketing lead.
Release planning	Estimate the number of iterations that can be performed before the release date.	Scrum master, business analyst, marketing lead.
Iteration planning	Provide more details on the story and estimate the hours to complete the associated tasks.	Scrum master, developer, business analyst, tester, marketing (whole team).
Daily standups	Meeting reviews the status of the iteration on a daily basis.	Scrum master, developer, business analyst, tester, marketing, managers, users (whole team).
Daily Scrum	The set of activities that are done on a daily basis.	Developer, tester, analyst, marketing.
Iteration retrospect	Review the units with stakeholders for their feedback.	Stakeholder, customer support, technical support, developer, tester, marketing (whole team). Management wanting status, user or user rep.
Release	User acceptance test, software bundle, documents, etc. Whole team is intact for the and during the release (release is a combination of n iterations).	Customer service, server support, Scrum master, developer, business analyst, tester, marketing, managers, users (whole team).

APPENDIX C: DEFINITIONS OF WELL-KNOWN CONTROL MECHANISMS

Table C-1: Definitions of Control

Control mode	Description
Behavioral	Behaviors that transform inputs to outputs are known (Kirsch, 1996); Rules and procedures articulated (Kirsch, 1997). Controller monitors and evaluates controllee's behavior (Kirsch, 1996). Explicit link exists between extrinsic rewards and following behaviors (Kirsch, 1996); Rewards based on following rules and procedures (Kirsch, 1997). The extent to which the manager monitors and evaluates team members' behavior in order

	to assist them (Henderson & Lee, 1992). Specifying behaviors for individuals to follow and then applying sanctions or regards based on their compliance with those behaviors (Haney, 2009). Evaluation when a task is taking place (Jaworski, 1988.
Outcome	Desired task outcomes are known and measurable (Kirsch, 1996); outcomes and goals articulated (Kirsch, 1997). Controller evaluates whether outcomes were met (Kirsch, 1996). Explicit link exists between extrinsic rewards and producing outcomes (Kirsch, 1996); Rewards based on producing outcomes and goals (Kirsch, 1997). Managerial outcome—the degree to which the manager monitors and evaluates only the outcome produced by the team members (Henderson and Lee ,1992). Team-member outcome—an attempt to influence the performance of the team by providing feedback on performance and goal-related outcomes (Henderson & Lee, 1992); e.g., structured walk-throughs; focuses only on the resulting design (artifact) not on the process by which the design is created. Specifying desired outcomes and rewarding or sanctioning individuals based on whether or not they attain the desired outcomes. Evaluation after a task (Jaworski, 1988). Outlining a set of project goals to be achieved; and rewards are made contingent on the accomplishment of goals (Maruping et al., 2009b). Emphasis is on software development team outputs (Henderson and Lee 1992, as cited in Maruping et al., 2009b).
Clan	Task-related behaviors and outcomes are not pre-specified (Kirsch, 1996). Goals are determined by clan and evolve during the task period (Kirsch, 1996); Specific task goals evolve over the life of the task (Kirsch, 1997). Clan identifies and reinforces acceptable behaviors (Kirsch, 1996); Identification and reinforcement of acceptable behaviors (Kirsch, 1997). Rewards are based on acting in accordance within clan's values and attitudes (Kirsch, 1996). Shared experiences, values, and beliefs among the clan members (Kirsch, 1996); Common values, beliefs, & problem-solving philosophy (Kirsch, 1997); norms and values established in various social units (Jaworski, 1988). Shared values center around what constitutes proper behavior (Haney, 2009). Can be implemented through rituals and ceremonies that reward those who share the attitudes and values of the clan (and that the clan believes will lead to success) (Haney, 2009). Norms and values internalized through a socialization process, eliminating the need for formal controls (Orlikowski, 1991). Members exhibit strong commitment to the clan (Kirsch, 1996). Socializing team members into a specific set of norms and values that are valued by the organization (Maruping et al., 2009b). Management may espouse the values, but the clan rewards or sanctions those behaviors (Ouchi, 1979).
Self	Controllee sets own task goals and procedures (Kirsch, 1996); individual defines task goals or procedures (Kirsch, 1997). Controllee is intrinsically motivated (Kirsch, 1996). Controllee engages in self-monitoring and self-evaluation (Kirsch, 1996); Individual monitors, rewards, and sanctions self (Kirsch, 1997). Reward are based partly on controlee's ability to self-manage (Kirsch, 1996); rewards based, in part, on individual's self-control skills (Kirsch, 1997). Team member self-control is the extent to which an individual exercises freedom or autonomy to determine both what actions are required and how to execute activities (as cited in Henderson & Lee, 1992). May be implemented when organizations cannot adequately measure behavioral performance or standardize transformation procedures. Individuals determine what actions are required and how to execute those actions (Henderson & Lee, 1992).

APPENDIX D: SAMPLE GUIDING INTERVIEW QUESTIONS

Table D-1: Sample Set of Guiding Interview Questions

We understand that you have recently used Agile methods for a project within your organization. Tell us about the Waterfall software development methods that you currently use elsewhere in your organization. What are the drawbacks of the Waterfall method?
Why was Agile implemented? What benefits have you seen from implementing Agile? What can Agile do and what can't it do?
How was the team composed for this project?
Tell us about your experience with Agile. What worked well? What didn't work so well?
Who holds the Agile team accountable? Were there people who put in more effort and did a better job because of visibility?
As part of managing your resources (if a manager), what do you give up or what do you gain by using an Agile approach?
Who loses out when Agile is implemented in a large organization? Who wins?
Agile groups presumably hold themselves accountable rather than a manager holding them accountable.... would you agree? Why or why not?
So what type of control do IT managers have in the Agile world? Does your current management know what you were doing during the iteration?
Did management understand the process for Agile? Was your manager aware of the steps of Agile?
Can you drive a project better using an Agile perspective? What is your perception?
How does introducing Agile affect the relationship between the different departments in an organization?

ABOUT THE AUTHORS

Lakshman Mahadevan is an Assistant Professor at Emporia State University. He completed his PhD in MIS at the Fogelman College of Business at the University of Memphis in the summer of 2014. His current research focuses on the software-development process, information security, enterprise architecture, technology efficacy and sourcing. His work has been published in the *eService Journal* and AMCIS proceedings. He has served as reviewer for AMCIS, HICSS and ICIS conferences. Previously, Lakshman was an Enterprise Architect for FedEx with 10 years of technical architecture solution delivery experience for over 150 projects in areas such as critical application/infrastructure upgrades, new implementations of applications and technologies, data center migrations with dual data center/disaster recovery needs.

William J. Kettinger is a Professor and the FedEx Endowed Chair in MIS at the Fogelman College of Business and Economics at the University of Memphis. He previously served as Professor and Moore Foundation Fellow at the Moore School of Business of the University of South Carolina. He teaches in both the masters and doctoral program in MIS at the University of Memphis and actively engages the Memphis business community, including such companies as FedEx, International Paper, and AutoZone. His research focuses on senior executives, strategic information management, IT and supply chain management, systems development, and online service quality. He has contributed to over 100 publications, including four books and 70+ refereed papers in such journals as *Information System Research, MIS Quarterly, JMIS, JAIS, ISJ, European Journal of IS, Decision Sciences, MIS Quarterly Executive,* and *Sloan Management Review*. He currently serves, or has served, as a senior editor of *MISQ* and *MISQE* and as associate editor of *MISQ, ISR*, and *Journal of the Association for Information Systems*.

Thomas O. Meservy is an Assistant Professor of Information Systems at Brigham Young University. He graduated with his PhD in Management from the University of Arizona in 2007 and earlier from Brigham Young University with a BS in Management and a Masters of Information Systems Management. His research interests include software development tools and methodologies, collaboration, and automated understanding of human nonverbal behavior. His work has been published in various journals such as *Information Systems Research*, *JMIS, Data Base, IEEE*

Computer, IEEE Intelligent Systems, IEEE Transactions on Intelligent Transportation Systems, Group Decision and Negotiation, and numerous proceedings of major IS conferences such as ICIS, HICSS, and AMCIS. Much of his research has been funded. He was an early proponent (late 1990's) of Agile methodologies. He also holds a number of professional technical certifications.

Communications of the Association for Information Systems

ISSN: 1529-3181