Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2007 Proceedings

Americas Conference on Information Systems (AMCIS)

December 2007

Analyzing Scrum Agile Software Development with Development Process, Social Factor, and Project Management Lenses

YongSeog Kim
USU

Analyzing Scrum Agile Software Development with Development Process, Social Factor, and Project Management Lenses

YongSeog Kim
Utah State University
yong.kim@usu.edu

ABSTRACT

Despite their popularity of agile software development methods, there have been very limited studies to match specific theories with particular aspects of agile software development practices. This study intends to provide a theoretical framework by matching three different types of analysis lenses—process management, social factor, project management—to each component of Scrum software development practices. Researchers can also use them to investigate how each component of Scrum is related each other. With three process management lenses (communication, coordination, and control), we first investigate how and where communication and coordination processes between different elements start, and how management teams can control them. In addition, social factor theories (social facilitation, social loafing, and group motivational gain) and theory of complex adaptive system are introduced to explain core practices of and highlight their implications in Scrum. Finally, we present several important issues on how to optimally manage Scrum teams, how to extend Scrum practices to large-scale and global software development, and how to incorporate pair programming practice from XP.

Keywords

Scrum, Communication, Coordination, Control, Social factor lenses, Project management lenses

INTRODUCTION

Recently, agile software development methods have garnered a lot of attention from researchers in academics and practitioners in the software engineering industry due to their lighter and faster development life cycle. Among many agile methods, the most well known and well studied method is eXtreme Programming (XP) that provides software developers with the best 12 engineering practices that cover all four major steps of software development life cycle from planning to implementation and testing (Beck, 2004). Recently, Scrum gains popularity from industry practitioners and academic researchers mainly because Scrum, in contrast to XP, is explicitly designed to provide guidance of managing and tracking software development projects (Schwaber & Beedle, 2002). While Scrum focuses on identifying any deficiencies or impediments in the development process, it is not constrained to the specific techniques and methods for the implementation process (Abrahamsson et al., 2003). All agile methods commonly emphasize incremental and iterative (small software releases and continuous updates) development (IID) practices (Fowler & Highsmith, 2001) to ensure the development of software in time, on budget, and of desired quality. Therefore, their success is heavily dependent on strong cooperation among engineers, strong customer collaboration, and proactive accommodation of last moment changes.

Many prior studies (Bowers et al., 2002; Cho et al., 2006; Mann & Maurer, 2005) disseminate successful case stories of agile methods in various development environments and share insights on "what" went well and "what" went wrong. In order to complete these case studies, what we need is theoretical and conceptual studies that provide answers to more fundamental questions, "why and how" a project with agile process leads to a successful or failed project. As one of such attempts, this study intends to provide a theoretical framework and possible theories for future research on Scrum. We note that several theories from organization science and communication are readily applicable to explain how a change in core agile elements (say, change in the diversity of software engineers) causes successive changes in other elements (say, trust, motivation, and productivity of each software engineer), affecting the overall outcomes of agile methods. In particular, it is one of our main objectives to discover and identify mappings that relate applicable theories to a specific set of relevant elements of Scrum agile practice. For this purpose, we subjectively select three different lenses—development process lenses, social factor lenses, and project management lenses to narrow down our focus while addressing most aspects of software development projects. The development process lenses focuses on control, communication, and coordination channels among major parties in the development process, while social factor lenses focuses on social factors that cause positive and negative impacts on

the performance of individual members and team itself. The project management issues include scalability, combination of agile methods, and conflict management in agile project team management. We graphically represent our research model in Figure 1.

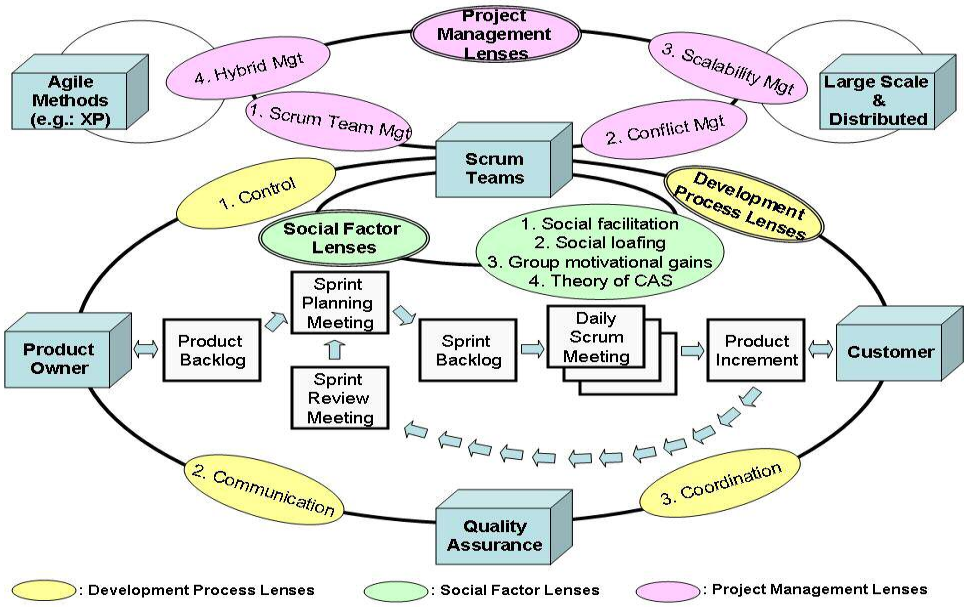

Figure 1: Research Model

The remainder of this paper is organized as follows. Agile software development with Scrum is first introduced with its elements. Next, we use three development process lenses (communication, coordination, and control) to study how Scrum supports each of development processes, how they are related each other, and how they affect the performance of Scrum. In the following section, we analyze Scrum practices from social factor theories (social facilitation, social loafing, and group motivational gain) and the theory of complex adaptive system to highlight their implications in Scrum practices. Next, we discuss key project management issues such as team management, conflict management, scalability management, and hybrid management in Scrum. Finally, we conclude the paper with some suggestions for possible future research.

AGILE SOFTWARE DEVELOPMENT WITH SCRUM

The Scrum method is a team-based lightweight process to manage software development using iterative and incremental practices (Schwaber & Beedle, 2002). The core element of Scrum is cross-functional self-managed Scrum teams that work effectively through continuous interactions among team members. Scrum teams are also self-organizing and fully autonomous, guided by their own knowledge and experience. Among the team members, Scrum Master (SM) is the person who does administrative work, such as arranging the daily Scrum meeting, removing any production impediments, and acquiring resources as needed. The SM also serves as a liaison between the team and other departments, and keeps track of what is going on in daily Scrum meeting to gauge the velocity of the team (i.e., how fast the team makes progress). The SM works with the customers and management to identify a Product Owner (PO) who is only person to manage and sustain the features and priority of a single product in a release to satisfy the market needs. Typically, the PO comes from the marketing department and acts as the product manager or project manager. Another important human resource in Scrum is Quality Assurance (QA) person. The QA person is responsible for assuring the quality of projects by working with the Scrum teams, and coaching them, rather than acting like a policeman checking long lists of quality issues (Vriens, 2003).

The Product Backlog (PB) encompasses all features, functions, technologies, enhancements, and bug fixes needed to be done (Schwaber & Beedle, 2002). The PB is initially incomplete with minimum requirement of the product, and evolves as additional requirements from various sources (i.e., customer requests, brainstorming sessions, and new technologies) emerge. The importance of the PB is in the fact that it lists all features in order of priority based on the relevance to the product. Other process elements of Scrum include several important meetings such as the daily Scrum meeting, the daily Scrum of Scrums meeting, the sprint planning meeting, and the sprint review meeting. The daily Scrum meeting is a short (usually 5-15 minutes) standup meeting in which developers talk about what has been done since last meeting, what will be done before the next meeting, and what impediments are in its way. Note that each development team may have different schedules for the

daily Scrum meeting. The daily Scrum of Scrums is a daily meeting for SMs from multiple Scrum teams. In addition to these informal daily meetings, there are two important meetings at the end of each thirty-day iteration, sprint. The sprint planning meeting is a sprint meeting where Scrum teams and PO work together to define as many items in the PB as possible they can implement within a sprint. The set of tasks to implement the selected items from the PB is entered into a sprint backlog, and developers check the progress by inspecting the product increment. In the sprint review meeting, developers demonstrate accomplishments to the stakeholders, product owners, customers, and other department representatives to receive constructive feedback, which helps shape and refine the PB. Typically, it takes a day to conduct the sprint review meeting and the sprint planning meeting.

DEVELOPMENT PROCESS LENSES IN SCRUM

Communication

The root cause of most software failures is ineffective communication (Parnas, 2006) and hence the formal and informal exchange of information is an essential process in all software development (Curtis et al., 1988). However, it is more critical in agile software development mainly because of the dynamic and evolutionary development process of agile methods. In Scrum, developers create and update the priority of tasks through interactions and discussions in the daily Scrum meeting and the sprint review meeting. In particular, the daily Scrum meeting is a good time for team members to understand what other members are working on and identify obstacles to overcome. Through good communication, team members are able to refine the goal for each sprint and improve the quality of products. It is also critical to maintain a frequent and efficient communication channel between Scrum teams and the customers through Scrum meetings. After several Scrum planning and review meetings, both the customers and the developers can have a better understanding of both the requirements and the limitations of the development process (Mann & Maurer, 2005). When the customers cannot attend to Scrum meetings, the company can have a customer representative visit the customers or maintain a Web conference to show the progress of the products and to obtain feedback from them. It is also necessary to keep an efficient communication channel between Scrum teams and the product owner who makes final saying regarding the priority of requirements based on the customer's feedback in sprint reviews (Schatz & Abdelshafi, 2005).

However, it is not clear how to operate daily Scrum meeting (standup versus seated meeting) for a short meeting time (less than 15 minutes). It is also arguable whether developers should have Scrum meeting everyday even when there are no specific agenda to discuss although these meetings can help developers visually remember what needs to be done on a daily basis. In order to improve the communication and interactions among team members, the company typically relocates all the scattered developers in the same team into the same place where two developers share cubicles. However, it would be possible for developers to be constantly distracted by cubicle partners who are often having conversations with other coworkers (Parrish et al., 2004).

Coordination

The second development process lenses is coordination, the set of activities to effectively manage interdependent tasks (Carmel & Agarwal, 2001). A well defined software process, a well documented software architecture, and detailed project planning in traditional software development process (e.g., waterfall methods) are good example of coordination mechanisms. However, many formal coordination mechanisms are replaced by frequent, intensive, informal communication among team members and with the customer in agile methods. However, minimal coordination mechanisms like coordination norms have been shown to be beneficial to increase group effectiveness, although they may lead to the reduced innovation and inertia to change (Ghosh et al., 2004). Note that coordination norms could be formed in four different ways: be explicitly mandated by a supervisor, emerge from critical events, develop through primacy, and be carry-overs from the past.

According to Ghosh et al. (2004), three different types of coordination norms—preventive, corrective, and adaptive—can be established at different times through different processes. For example, preventive norms may be established at the initial stages through deliberate discussion and prior work experiences. In contrast, corrective and adaptive norms may be generated as team members respond to an unexpected event (preventive norms) or coordinate collectively and effectively over time by sharing each other's preferences, rhythms, and work styles. Another study shows how coordination meetings helped work teams generate longer-term stability (Maznevski & Chudoba, 2000). A good example of coordination norms can be found in Schatz & Abdelshafi (2005) that introduces criteria on what constitutes a completed feature to be shown to stakeholders at sprint reviews. These coordination norms leave no room for misunderstanding about what a completed feature means, making a progress checking of products consistent across Scrum teams as well as enhancing coordination among team members. Having a coding standard to keep consistency in coding/code inspection in XP practices is another good example for coordination.

Control

The third development process lenses is control, the process of adhering to goals, policies, standards, or quality levels (Carmel & Agarwal, 2001). The control process concerns project management and reporting mechanisms to ensure the development of software in time, on budget, and of desired quality (Holmström et al., 2006). In Scrum, the developers have control over how and when development is completed, and update the priority of tasks through discussions in the daily Scrum meeting and the sprint review meeting. Note, however, that developers are not often able to fully consider all the dependencies and interconnections among modules because of their myopic planning and design in each sprint planning meeting, resulting in inconsistent product outputs across team members within the same Scrum team (Cho et al., 2006). In particular, when developers have to complete a set of tasks, they tend to complete tasks in a quick and dirty way rather than to think of how each task will be flexible enough for future needs. Further, it is difficult to keep the output of product consistent across teams due to the limited communication between Scrum teams. For example, several teams may simultaneously work on "look and feel" design that should be implemented by a designated team (e.g., an architecture team). This implies that there should be a person (possibly a SM in each team) who is responsible for checking consistency of products across Scrum teams.

Communication, coordination, and control in Scrum

Product backlog is at the core of communication, coordination, and control process in Scrum. After considering the prioritized list of all business and technical functionality of the product, Scrum teams determine a set of tasks to implement in the next sprint, and record them into a sprint backlog. In short, a product backlog (for the whole project period) and a sprint log (for a period of a sprint) are valuable means of communication, coordination, and control activities. Therefore, it is recommended that both a product backlog and a spring log be highly visible to all the people in the project. The daily Scrum meeting and daily Scrum of Scrums meeting are mainly for keeping efficient communication channels between team members in the same Scrum team and across Scrum teams, respectively. Therefore, the main purpose of these meetings is to communicate what is going in each Scrum team and across multiple Scrum teams, not necessarily to find complete solutions to the impediments. Note, however, that daily scrum meetings can be used as a useful way to remind developers what need to be done and monitor the progress of the projects on a daily basis.

In contrast, both sprint review and planning meeting provide a mechanism to control and coordinate efforts of Scrum team members. For example, the product owner checks in sprint review meeting what has been done, what things need to be improved, and what things the team has done well. Based on her observation, the product owner may change the priority of product requirements and communicate directly with developers. Although Scrum does not allow the stakeholders to directly control the product backlog, stakeholders can provide inputs during monthly sprint reviews (Schatz & Abdelshafi, 2005). However, Scrum can make it difficult to determine how much work remained on specific features or how much work remained until the release because, by definition, the requirements change from sprint to sprint. Therefore, the teams should work closely with the product owners to understand the business value of the features they were implementing. Teams also need to build common and standardized processes at the initiation of the project to make changes later in the project at a lower cost and in a more coordinated manner (Lee et al., 2006). For example, they can have a portfolio of various communication and collaboration technology options from which they can choose based on what works or doesn't work for a particular stage of the project. While building and maintaining flexibility is critical, successful software teams also exhibit disciplined adherence to the agreed-upon strategies and processes.

SOCIAL FACTOR LENSES IN SCRUM

Social Facilitation, Social Loafing, and Group Motivational Gain

According to the theory of social facilitation, the presence of other people positively affects the performance of each individual on a simple task, but negatively on a difficult or new task (Aiello & Douthitt, 2001). Several studies have explained the social facilitation effects with one of following theories: drive theory, enhanced drive in the presence of another individual improves simple task performance; evaluation apprehension theory, possibility of evaluation, not mere presence, is drive inducing; and cognitive theory, performance improves up to a distraction point. Another social factor, social loafing negatively affects the performance effort and performance itself of each individual. Social loafing in groups is partly attributed to members' feeling that their contributions are not essential and it is not easy to accurately extract their contribution from others (Karau & Williams, 1993). Further, people typically expect others to slack off in group work and hence tend to reduce their efforts to maintain equity. It is possible to eliminate social loafing effects by making individuals' contributions verifiable (Harkins & Jackson, 1985). In contrast, group motivational gain is the tendency that people increase their effort to help co-workers when working in groups. Group motivational gains may be explained by a social

compensation effect—the more able partners increase their effort to help the less able partners (Williams & Karau, 1991), but decrease their efforts when the partner has high abilities (Hart et al., 2001). However, motivational gains can be driven by the less able partners, when their contribution holds the key to the group performance (Witte, 1989). According to a Kohler effect, when team members have moderate differences in abilities (not too similar or dissimilar), they perform better in a group compared to their expected individual performance (Witte, 1989).

A recent case study (Cho et al., 2006) reports that social facilitation effect and group motivational gains reduce development times and lowered bug rates. In their study, the company also holds a sprint review meeting called a "products fair" to discourage social loafing. The products fair is unique in the sense that developers in all teams, QA personnel, and people in other departments (sales, marketing, and customer support) attend to the meeting and developers show their accomplishments during the sprint. This causes evaluation apprehension (Cottrell, 1972) mainly because all developers have to show what they have done and they are concerned about how they are evaluated by other people. Group motivational gains were also observed when a team member helped other team members who were not familiar with new development tools or programming languages.

Theory of Complex Adaptive System in Scrum

Complex adaptive systems (CAS) are open systems that change and learn from experience and dynamic interactions with their environment (Cilliers, 2000). The theory of CAS provides us a framework to understand agile practices and Scrum team. The first implication of the theory of CAS is that, in order to build a complex system in dynamic environment, one must build it by incremental and iterative principle mainly because it is not possible to plan the entire system first and then build it in rapidly changing environment. Practices of working iteratively and making small increments can be beneficial for both customers and programmers by decreasing stress and increasing confidence. The customers appreciate the early delivery of business value, while the programmers value the early feedback from the customer (Vriens, 2003). The theory of CAS also implies that any changes to the project requirements and technology should be proactively handled. Note that, according to the theory of CAS, unanticipated new requirements of a new system are not an indication of malfunctioning, but an output of learning and adaptation. Scrum team is a complex adaptive system in which all members learn from their environment and experience how to do what is needed. Scrum team consists of members who work closely together and have a high degree of adaptive capacity and flexibility to incorporate any last minute changes.

It is important to note that the theory of CAS and agile principles are far from undisciplined development practices. The key difference between agile principles and undisciplined development practices is that agile methods limit the scope of the planning process to the current iteration and generate the next set of plans only after executing and evaluating the outcome of current plan (Martin, 2000). Cohn (2005) argues that acknowledging the impossibility of creating a perfect plan and planning at three different levels—the current day, the sprint, and the release—are what make Scrum successful. That is, Scrum team makes the daily plan fairly precise, while keeping the release plan the least precise of all plans containing only a prioritized list of desired functionality. Evolving plans created for different reasons at different levels also help the team view the project from both the microscopic and the macroscopic perspectives, and help the team keep the opportunity to synchronize the plan with reality as the team acquires new knowledge and technology (Cohn, 2005).

PROJECT MANAGEMENT LENSES IN SCRUM

In this section, we focus on future research direction from the perspectives of Scrum team management, conflict management, scalability management, and hybrid management.

Scrum Team Management

One of the main objectives of Scrum team management is to provide a guideline on how to organize and maintain each Scrum team of developers with various levels of the knowledge and skills. In terms of the size of the team, most Scrum teams are operated by small-size teams (i.e., between 5 and 9 members) and a large number of developers are divided into "multiple teams while minimizing the dependencies between the teams but maximizing the cohesion of the work between them" (Schwaber & Beedle, 2002). With multiple Scrum teams, it will be necessary to arrange a meeting among SMs, called daily Scrum of Scrums meeting to improve the cohesion of the work. To maximize the coordination and communication across teams, each team is also responsible for conforming to any existing standards, conventions, architectures, and technology. Scrum teams are cross-functional and hence are responsible for all tasks including analysis, design, implementation, and user documentation. Therefore, it is important to assign appropriate management and professional roles each QA, SM, and PO should take. For example, the PO should be only official person responsible for prioritizing requirements although the whole

team may involve in brainstorming about project plans and implementations (Cohn, 2005). Further, developers, QAs, and SMs should be assigned to each team so that it can operate independently.

Note that the company can maintain homogeneous or heterogeneous teams based on the degree of homogeneity (or diversity) of team members. The homogeneity of team members can be measured in terms of their backgrounds (e.g., age and gender), personalities (introvert vs. extravert, leader-type vs. follower-type, optimistic vs. pessimistic, and group player vs. individual contributor), and development skills (e.g., experienced vs. inexperienced in programming languages). According to several studies (Cox et al., 1991; Milliken and Martins, 1996), the diversity of teams positively affects the overall team performance and the quality of group decisions. It is easy to visualize that less experienced developers in business logic and programming skills quickly learn from experienced developers in the same Scrum team. However, other studies (O'Reilly et al., 1989) report that the extreme heterogeneity of teams deteriorates the overall team performance because team members do not feel "team spirit" and do not collaborate with other members. According to social identity/deidentity (SIDE) theory (Lea & Spears, 1991), people experience feelings of isolation and anonymity when they communicate with others that they do not know well. Other studies also state that the diversity should be taken carefully to avoid the possible damaging effects (Lau & Murnighan, 1998).

Despite many studies with opposite findings and suggestions, the diversity of team members within a team is currently well recognized as a powerful management practice mainly because heterogeneous teams are expected to bring a variety of skills and perspectives to solve problems and implement the solutions (Beck, 2004). However, researchers have not paid much attention to optimal allocation of human resources across Scrum teams. For example, it is important to investigate whether the diversity across teams (e.g., operating multiple Scrum teams with different competency levels) is beneficial to the overall project performance. It will be interesting to see if a Scrum practice in which "dream" teams with superior programmers work on tasks with top priority and other teams work on tasks with lower priority can produce better project outcomes than another Scrum practice in which identical teams in terms of programmer skills work tasks in sequential order of the product backlog.

Conflict Management

A conflict resolution mechanism is defined as a preferred procedure that people adopt to bring a conflict situation to a settlement. Five most common patterns of conflict management (Rahim, 1983) in group settings are: avoidance style (avoiding the conflict situation), accommodating style (focusing on areas of agreement and thus smoothing over differences), competition style (forcing one's own views on others), collaborative style (integrating the views of all involved), compromise style (finding a middle ground solution). Collaborative style may occur when members focus on finding a common solution that addresses everyone's interest. Follett (1940) suggests that other styles of conflict management (e.g., compromise) deal with what already exists, whereas integration style (e.g., collaboration) creates something new. It is also noted that using a collaborative style stabilizes the conflict in the group, but in a compromise style the conflict is likely to recur in other forms (Follett, 1940). According to Miranda and Bostrom (1993), groups using computer support in decision-making are likely to have less affective conflict, but have more task conflict and greater levels of collaborative conflict resolution than groups that meet face-to-face. These behaviors or attitudes may be more evident in virtual team's environment due to the physical separation of the members and the technology-supported interaction.

Conflict resolution mechanism is closely tied to individual characteristics of the team members. Paul et al. (2005) note that the individualistic-collectivistic orientation of the team members influences the level of collaborative style pursued by a team, highlighting the importance of motivating team members to collaborate in group work. Several studies have examined the utility of maintaining shared mutual knowledge (Cramton, 2001) and shared mental models (Espinosa et al., 2001) on conflict resolution within teams. Research on the diversity of teams has addressed various relevant issues, such as trust among the team members (Jarvenpaa et al., 1998), conflict and collaborative style and their effects on decision-making of virtual teams (Montoya-Weiss et al, 2001), and effective leadership (Kayworth & Leidner, 2001). In conflict management, performance measures are based on more qualitative rather than quantitative measures such as the level of agreement among members and their perceptions of participation, decision quality, and satisfaction with the decision process.

Scalability Management

The scalability management of Scrum addresses the question of whether Scrum can be applied to large-scale and mission critical projects or whether Scrum can be extended to work environment in which Scrum teams are geographically separated from each other (Reifer et al., 2003). Note that the distributed development environment significantly increase the complexity of software development activities (i.e. communication, coordination, and control), requiring conventional core agile practices to be modified with more rigor and disciplines using coordination routines and norms (Carmel & Agarwal, 2001;

Montoya-Weiss et al., 2001; Parnas, 2006). Remember that the core principle of agile software developments is to value working software over comprehensive documentation, informal customer collaboration over formal contract negotiation, responding to change over following a plan, tacit knowledge over explicit knowledge (Nerur et al., 2005). Several communication and organization theorists (Cramton, 2001; Maznevski & Chudoba, 2000) also claim that comprehensive documentation, explicit mutual knowledge, knowledge sharing and trust, and formal communication become more critical when geographical separation, cultural differences, language barriers, and organizational boundaries hamper efficient and effective communication and coordination.

A good example of agile development practice with both practices of flexibility and rigor can be found in Drobka et al. (2004). In their case study, developers use structural (e.g., class diagrams), functional (e.g., use cases), and procedural view of the system to communicate architectural view of the systems during the project's early iterations. At the same time, developers minimize their emphasis on other documents, while keeping clean and simple source code to effectively and efficiently respond to changes and reduce the cost of moving information between people (Jain & Meso, 2004). The real challenge in scalability management for agile methods including Scrum is "not to find ways to avoid documenting, but to find ways to produce useful documents—documents that take time but save more time" (Parnas, 2006). Note that new added-on norms of rigor and documentation should not interfere with the ultimate paradigm of "agile" software development process. In short, agility and flexibility in pure agile methods should be partnered with disciplined adherence to the agreed-upon strategies and processes to achieve success in global software development projects (Lee et al., 2006).

Hybrid Management with Other Agile Practices

The hybrid management of Scrum with other agile methods addresses the question of how well Scrum can adopt practices from other agile methods. Note that Scrum provides guidance of managing and tracking projects, but does not provide any specific techniques for the implementation process. Therefore, the possibility of using of Scrum as a wrapper for other agile methods such as XP has been suggested (Schwaber & Beedle, 2002; Vriens, 2003) because XP can complete Scrum by providing a set of engineering practices for implementation and testing. In particular, pair programming in XP is one of the most well studied practices, in which two programmers collaboratively work at the computer terminal taking turns with the keyboard, one acting as the driver who writes codes and the other as the navigator who inspects the code and thinks about the program logic. The question is whether it is worthy to incorporate pair programming practice into Scrum and, if it is worthy, in which way we should incorporate pair programming practice into Scrum.

Several studies report positive outcomes of pair programming such as higher quality code, higher satisfaction among the developers, and knowledge transfer (Beck, 2000; Balijepally, 2005). However, the overall performance of Scrum teams can be deteriorated by putting a shy developer with someone who is overbearing or pairing up inexperienced developers together (Drobka et al., 2004). Even when we want to pair experienced developers with less experienced developers, it may not be possible to do so because experienced developers are so busy that they cannot find three to four hours of uninterrupted time each day to work together. Also note that pair programming does not always work for all tasks. For example, it is shown that pair programming does not work for simple, well understood tasks, but works better for intellective and disjunctive tasks (e.g., selecting one of best solutions) because the team performance is determined by the best group member in these tasks (Laughlin et al., 2003). However, in conjunctive tasks as in assembly line, pair programming is not a good choice because the team performance is limited by the worst performing member. Therefore, to bring practices such as pair programming into Scrum, Scrum masters begin with detailed task analysis to determine whether there are sufficient intellective and disjunctive components in projects to benefit from pair programming practices (Balijepally, 2005).

CONCLUSION AND FUTURE RESEARCH

The main contribution of this study is to provide a theoretical framework and possible theories for future research on Scrum by matching three different types of analysis lenses—process management, social factor, project management—with particular aspects of Scrum software development practices. This study investigates how and where a communication process between the Scrum teams and the customers starts, how it affects the overall performance of Scrum teams, and what mechanisms management and Scrum teams utilize to control communication process. In addition, three social factor theories (social facilitation, social loafing, and group motivational gain) and theory of CAS are applied to explain core practices in Scrum agile methods. We also present several important issues such as conflict management, scalability management, and hybrid management with other agile practices. This study serves as a first step to find a solution to the more fundamental research question: how can we optimize and how should we apply Scrum agile methods to large-scale and mission critical software development project? For example, a future research can develop strategies and work environment that discourage social loafing, but encourage social facilitation and group motivational gain. Another interesting future research will be a case study to investigate hybrid methodologies tailored for the specific environment. Currently, we are collecting data from a

local company that uses Scrum as a wrapper for several XP practices such as pair programming, collective code ownership, minimal documentation, and collective code ownership.

REFERENCES

1. Abrahamsson, P., Warsta, J., Siponen, M., and Ronkainen, J. "New Directions on Agile Methods: A Comparative Analysis," *Proceedings of the 25th International Conference on Software Engineering*, 2003, pp. 244–254.
2. Aiello, J. R., and Douthitt, E. A. "Social Facilitation from Triplett to Electronic Performance Monitoring," *Group Dynamics* (5:3), 2001, pp. 163-180.
3. Beck, K. and Andres, C. Extreme Programming Explained, 2nd edition. Addison-Wesley, Reading, MA, 2004.
4. Balijepally, V. "Collaborative Software Development in Agile Methodologies–Perspectives from Small Group Research," *Proceedings of the 11th Americas Conference on Information Systems*, 2005, pp. 3138-3146.
5. Bowers, J., May J., Melander, E., Baarman, M., and Ayoob, A. "Tailoring XP for Large Systems Mission Critical Software Development," *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods – XP/Agile Universe 2002,* 2002, pp. 100-111.
6. Cilliers, P. "What Can We Learn From a Theory of Complexity?" *Emergence* (2:1), 2000, pp. 23-33.
7. Carmel, E., and Agarwal, R. "Tactical Approaches for Alleviating Distance in Global Software Development," *IEEE Software* (18:2), 2001, pp. 22–29.
8. Cho, J., Kim, Y., and Olsen, D. "A case study on the applicability and effectiveness of Scrum in mission critical and large-scale projects," *Proceedings of the 12th Americas Conference on Information Systems*, 2006, pp. 3705–3711.
9. Cohn, M. Agile Estimating and Planning, Prentice Hall, 2005.
10. Cottrell, N.B. "Social Facilitation," in C.G. McClintock (Ed.) *Experimental Social Psychology,* Holt, New York, 1972, pp. 185-236.
11. Cox, T., Lobel, S.A., and McLeod, P.L. "Effects of Ethnic Group Cultural Differences on Cooperative and Competitive Behavior on a Group Task," *Academy of Management Journal* (34:4), 1991, pp. 827-847.
12. Cramton, C.D. "The mutual knowledge problem and its consequences for dispersed collaboration," *Organization Science* (12:3), 2001, pp. 346-371.
13. Drobka, J., Noftz, D., and Raghu, R. "Piloting XP on Four Mission-Critical Projects," *IEEE Software*, 2004, pp. 70-75.
14. Espinosa, J.A., Kraut, R.E., Slaughter, S.A., Lerch, J.F., Herbsleb, J,D., and Mockus, A. "Shared mental models and coordination in large-scale, distributed software development," *Proceedings of the International Conference on Information Systems*, 2001, pp. 425-423.
15. Follet, M.P. "Constructive conflict," In H,C, Metcalf and L, Urwick (eds,). Dynamic Administration: The Collected Papers of Mary Parker Follett. New York: Harper & Brothers, 1940, pp. 30-49.
16. Fowler, M. and Highsmith, J. "Agile methodologists agree on something," *Software Development* (9), 2001, pp. 28–32.
17. Ghosh, T., Yates, A., and Orlikowski, W. J. "Using communication norms for coordination: Evidence from a distributed team," *Proceedings of 25th International Conference on Information Systems*, 2004, pp. 115-128.
18. Harkins, S.G. and Jackson, J.M. "The role of evaluation in elimination social loafing," *Personality and Social Psychology* (11), 1985, pp. 575-584.
19. Hart, J.W., Bridgett, K.J., and Karau, S.J. "Coworker Ability and Effort as Determinants of Individual Effort on a Collective Task," *Group Dynamics* (5:3), 2001, pp. 181-190.
20. Jain, R. and Meso, P. "Theory of complex adaptive systems and agile software development," *Proceedings of the 10th Americas Conference on Information Systems*, New York, New York, August 2004, pp. 1661-1668.
21. Jarvenpaa, S.L, Knoll, K., and Leidner, D. "Is anybody out there? Antecedents of trust in global virtual teams," *Journal of Management Information Systems* (14:4), 1998, pp. 29-64.
22. Karau, S.J. and Williams, K.D. "Social Loafing: A Meta-Analytic Review and Theoretical Integration," *Journal of Personality and Social Psychology* (65:4), 1993, pp. 681-706.
23. Kayworth, T.R. and Leidner, D. "Leadership effectiveness in global virtual teams," *Journal of Management Information Systems* (18:3), 2001, pp. 7-40.

24. Lau, D.C. and Murnighan, J.K. "Demographic diversity and faultlines: The compositional dynamics of organizational groups," *Academy of Management Review,* (23:2), 1998, pp. 325-340.
25. Laughlin, P.R., Zander, M.L., Knievel, E.M., and Tan, T.K. "Groups Perform Better Than the Best Individuals on Letters-to-Numbers Problems: Informative Equations and Effective Strategies," *Journal of Personality and Social Psychology* (85:4), 2003, pp. 684-694.
26. Lee, G., Delone, W., and Espinosa, J.A. "Ambidextrous coping strategies in globally distributed software development projects," *Communications of the ACM* (49:10), 2006, pp. 35-40.
27. Lea, M. and Spears, R. "Computer-mediated communication, deindividuation and group decision-marking," *International Journal of Man-Machine Studies* (34), 1991, pp. 283-301.
28. Martin, R. "eXtreme Programming Development through Dialog," *IEEE Software* (17:4), 2000, pp. 12-13.
29. Mann, C. and Maurer, F. "A Case Study on the Impact of Scrum on Overtime and Customer Satisfaction," *Proceedings of the Agile Development Conference (ADC'05)*, 2005, pp 70-79.
30. Maznevski, M.L., and Chudoba, K. M. "Bridging space over time: Global virtual team dynamics and effectiveness," *Organization Science* (11:5), 2000, pp. 473-392.
31. Milliken, F.J., and Martins, L.L. "Searching for common threads: Understanding the multiple effects of diversity of organizational groups," *American Management Review* (21:2), 1996, pp. 402-433.
32. Miranda, S.M., and Bostrom, R.P. "The impact of group support systems on group conflict and conflict management," *Joumal of Management Information Systems* (10:3), 1993, pp. 63-95.
33. Montoya-Weiss, M.M., Massey, A.P., and Song, M. "Getting it together: Temporal coordination and conflict management in global virtual teams," *Academy of Management Journal* (44:6), 2001, pp. 1251-1262.
34. Nerur, S., Mahapatra, R., and Mangalaraj, G. "Challenges of migrating to agile methodologies," *Communication of the ACM* (48:5), 2005, pp. 73–78.
35. O'Reilly, C.A., Caldwell, D.F., and Barnett, W.P. "Work group demography, social integration and turnover," *Administrative Science Quarterly*, (34:1), 1989, pp. 21-37.
36. Paul, S., Samarah, I. M., Seetharaman, S. and Mykytyn, P. P. "An empirical investigation of collaborative conflict management style in group support system-based global virtual teams," *Journal of Management Information Systems* (21:3), 2005, pp. 185-222.
37. Parrish, A., Smith, R., Hale, D. and Hale, J. "A field study of developer pairs: Productivity impacts and Implications," *IEEE Software* (21:5), 2004, pp. 76-79.
38. Parnas, P. "Agile methods and GSD: The wrong solution to an old but real problem," *Communications of the ACM* (49:10), 2006, p. 29.
39. Rahim, M.A. "A measure of styles of handling interpersonal conflict," *Academy of Management Journal* (26:2), 1983, pp. 368-376.
40. Reifer, D.J., Maurer, F., and Erdogmus, H. "Scaling agile methods," *IEEE Software* (20:4), 2003, pp. 12–14.
41. Schwaber, K. and Beedle, M. "Agile software development with Scrum," Prentice Hall, Upper Saddle River, NJ, 2002.
42. Schatz, B. and Abdelshafi, I. "Primavera gets agile: A successful transition to agile development," *IEEE Software*, 2005, pp. 36-42.
43. Vriens, C. "Certifying for CMM Level 2 and ISO9001 with XP@Scrum," *Proceedings of the Agile Development Conference (ADC'03)*, 2003.
44. Williams, K.D., and Karau, S.J. "Social Loafing and Social Compensation: The Effects of Expectations of Co-Worker Performance," *Journal of Personality and Social Psychology* (61:4), 1991, pp. 570-581.
45. Witte, E.H. "Kohler Rediscovered: the Anti-Ringelmann Effect," *European Journal of Social Psychology* (19:2), 1989, pp. 147-154.