Issues in Information Systems

Volume XII, No. 1, pp. 213-223, 2011

MANAGEMENT GUIDELINES FOR SCRUM AGILE SOFTWARE DEVELOPMENT PROCESS

Juyun Cho, Colorado State University – Pueblo, joey.cho@colostate-pueblo.edu
Richard A. Huff, Colorado State University – Pueblo, rick.huff@colostate-pueblo.edu
David Olsen, Utah State University, david.olsen@usu.edu

ABSTRACT

This paper explores critical issues and challenges that might arise in Scrum agile software development processes and provides management guidelines to help organizations avoid and overcome barriers in adopting the Scrum method as a future software development method. A qualitative research method design was used to capture the knowledge of practitioners and scrutinize the Scrum software development process in its natural settings. An in-depth case study was conducted in two organizations where the Scrum method was fully integrated in every aspect of two organizations' software development processes. One organization provides large-scale and mission-critical applications and the other provides small- and medium-scale applications. Differences between two organizations provided useful contrasts for the data analysis.

Keywords: agile software development method, Scrum method, management guidelines, qualitative research method.

INTRODUCTION

Over the past four decades, many software development methods have been created and utilized in the software industry. Each method has different features and characteristics that distinguish it from other methods; in general, these methods can be classified as either a heavyweight or a lightweight method. The heavyweight methods, also considered traditional methods, usually focus on comprehensive planning, heavy documentation, and big design up-front. In contrast, the lightweight methods, also known as agile methods, concentrate 1) more on individuals and interactions than processes and tools, 2) more on working software than comprehensive documentation, 3) value customer collaboration more than contract negotiation, and 4) focus more on responding to change than following a plan (http://agilemanifesto.org).

The traditional methods are still widely used in the software industry because of their straightforward, methodical, and structured nature [13]; they have proved that they can provide high assurance, stability, and predictability [6]. However, they have a number of key shortcomings, including slow adaptation to constantly changing business requirements and a tendency to be over budget and/or behind schedule, delivering fewer features and functions than specified in the requirements [5, 6, 8, 26, 33, 36]. The need for a complete set of requirements prior to design is also a major challenge for the traditional methods due to vague user specifications.

As a remedy for the shortcomings of the traditional methods, agile software development methods, including Scrum, eXtreme Programming (XP), Crystal, and Adaptive Software Development (ASD), have been created [17] and evolved by practitioners since the 1990s; they are designed to embrace, rather than reject, high rates of change [2]. These new approaches focus mainly on iterative and incremental development, customer collaboration, and frequent delivery through a light and fast development cycle. Many researchers have reported that agile methods have the potential to provide a higher level of customer satisfaction, lower bug rates, a shorter development cycle, and a quicker adaptation to rapidly changing business requirements [5, 6, 24].

In spite of the potential benefits of the agile methods, many organizations are reluctant to throw their traditional methods away and jump into agile methods. Their reluctance is the result of several issues, including: 1) the agile methods significantly reduce the amount of documentation and rely heavily on tacit knowledge, 2) these methods have not been sufficiently tested for mission/safety-critical projects, 3) belief that these methods are not adequate for

highly stable projects, 4) a concern that agile methods can be successful only with talented individuals who favor many degrees of freedom, and 5) that agile methods are not appropriate for large-scale projects.

Although many positive benefits of the agile methods have been published, there have been few empirical field studies on the negative aspects of various agile methods. The negative aspects of the agile methods mentioned earlier imply that there are issues, problems, and challenges faced in developing high-quality software products using these methods. Identifying the issues, problems, and challenges of the agile methods should be more beneficial to organizations considering them than merely showing their positive benefits. Organizations can learn more lessons from the examining the negative aspects and learning how to avoid the obstacles in adopting the agile methods. Therefore, it is worthwhile to conduct a research project to identify the issues and challenges of agile methods and provide management guidelines.

An exploratory research process using observations, surveys, documentation, and interviews was conducted at two organizations. The contribution of this research is four-fold: 1) it identifies critical issues and challenges that may affect the quality of the application of agile methods, 2) it illustrates how agile methods can be adopted and utilized to effectively support the development of small-scale, large-scale, and mission-critical projects, 3) it provides lessons for using Scrum obtained from the field to assist Scrum practitioners, and 4) it provides management guidelines to help many organizations avoid and overcome obstacles when adopting the Scrum method as a future software development method.

RESEARCH BACKGROUND

For this research, two research sites were chosen for an in-depth case study. Both organizations have produced several high-quality software products through an agile software development methodology with one particular method, called Scrum. The rationale for the selection of the Scrum method among the other available agile methods was 1) Scrum is a widely used agile method in the software industry, in particular in the United States [21, 37], 2) Scrum is powerful and easy to learn [38], and 3) the Scrum method claims to be suitable to any size of project [27]. This study also investigated the framework (roles, ceremonies, and artifacts) and the empirical process (visibility, inspection, and adaptation) of the Scrum method in both small- and large-scale projects.

Two research sites were selected for their similarities as well as their differences, based on the technique of theoretical sampling [15]. Both organizations chosen for this study have used Scrum for the past few years in their systems development processes. The Scrum method in both organizations was integrated in every aspect of their software development processes, including planning, analysis, design, coding, and testing. While one organization has been providing large-scale and mission-critical applications, the other organization has been providing small- and medium-scale applications. The two organizations also differ on size, industry, and location. These differences between the organizations provide useful contrasts to be made during data analysis. One organization produces large-scale and mission-critical applications will be called the ABC firm, while the other organization will be called the XYZ firm.

The ABC firm has been providing mission-critical public safety software since 1978. The XYZ firm has been providing internet-based database applications primarily to clients in the government sector for over fifteen years. The ABC and The XYZ operate in different markets. The duration of a project at ABC firm is longer than one at XYZ, with the average duration of projects at ABC being 1-2 year(s) and at XYZ 3-6 months. ABC utilizes various computer programming languages, such as Java, C, C++, C#, and Perl, on both UNIX and Windows operating systems. XYZ mainly uses Java and HTML-based web programming languages on the Windows platform.
Table 1 summarizes various differences between the firms.

Table 1. Differences between ABC and XYZ firm

Category	ABC Firm	XYZ Firm
Main Applications	Jail Management System, Computer Aided Dispatch System Fire/Emergency Medical Service System Records Management System	Vital Statistics/ Record System Environmental Water Service System Human Service System
Size of Projects	Large-Scale	Small/Medium -Scale
Mission-critical Projects	All applications are mission-critical	Some of applications are mission-critical
Category	ABC Firm	XYZ Firm
Average Duration of Projects	1-2 year(s)	3-6 months
Computer Languages	Java, C, C++, C#, Perl	Java, HTML
Development Platform	Unix and Windows System	Windows System

RESEARCH METHODOLOGY

As mentioned in the introduction section, two in-depth case studies were employed as a research method for two primary reasons. First, a case study has the capability of scrutinizing a phenomenon in its natural settings and utilizing multiple data collection methods to gather information from one or more people, groups, or organizations [4, 7, 20, 30, 39]. Second, a case study is known to be a well-suited method for capturing the knowledge of practitioners and developing theories from it [3]. In an attempt to gather data, three types of data were collected from both firms to triangulate findings and enhance trustworthiness [14, 16]. First, observations of software development process were conducted through on-site visits and field notes were taken during the observation to make the strange familiar [12]. Second, an email survey was developed and conducted among software developers, QA personnel, and managers. The survey instrument was mainly used to refine interview questions. Finally, a formal face-to-face interview was conducted with executive officers, project managers, lead software engineers, and developers. All of the formal interviews were audio-taped, transcribed, and later coded for analysis. This triangulation in the process of data collection provides more useful information and different perspectives on the issues, allows for cross-checking, and yields stronger substantiation of constructs [10, 15, 25].

The first field study was conducted within the ABC firm. During the field study, which lasted for about 8 months, daily systems development activities under the Scum process, such as the daily Scrum meeting, the Scrum of Scrums meeting, the Sprint planning meeting, and the Sprint review meeting, were observed. In addition to the observation, an email survey was conducted among 15 people including developers, Quality Assurance (QA) personnel, and project managers. Some of selected project team members, including developers, lead engineers, project managers, and executive officers were interviewed both informally and formally. Each informal interview took an average of a half an hour and summary notes were taken. The formal interviews took an average of one hour and were audio-taped and transcribed later.

The second field study was conducted within the XYZ firm. The field study, which started with an informal interview with an executive officer, took about 4 months. The Scrum processes in 5 Scrum teams were observed and an email survey was conducted among developers, QA personnel, and project managers. The same survey questions used at ABC were utilized. Informal and formal interviews were conducted among 10 people including developers, project managers, a director of operations, and a senior vice president of operations.

At each site, the observation was conducted in both observe-only mode and in a mode where a participant observer was allowed to talk. Various levels of individuals, such as developers, lead engineers, project managers, and executive officers, were selected to provide data from multiple levels and perspectives. This was done to answer Leonard-Barton [22, p. 249], "In order to understand all the interacting factors, it is necessary that the research methodology slice vertically through organization, obtaining data from multiple levels and perspectives." While the primary unit of analysis was the Scrum software development process, the collection of inter-related data at other levels of analysis [25, 39, 40] was considered.

In the process of data analysis, grounded theory [15, 23, 35] was employed with an aim of generating descriptive and explanatory theory associated with Scrum software development process. This approach has been effectively utilized in organizational research [1, 11, 18, 19, 25, 34]. Grounded theory is "an approach to theory development that involves deriving constructs and laws directly from the immediate data that the researcher has collected rather than drawing on an existing theory" [14, p. 626]. The main purpose of grounded theory, which was initially developed by Strauss and Corbin [31], is to "demonstrate relations between conceptual categories and to specify the conditions under which theoretical relationships emerge, change, or are maintained" [9, p. 675].

RESEARCH RESULTS

Table 2 lists the categories and concepts found in the data analysis process. As shown in the table, the factors of structured development process and environment were constructed as a final category.

Table 2. Categories and concepts found in ABC and XYZ firm

Category	Concepts related ABC Firm	Concepts related to XYZ Firm
Structured Development Process	• Scrum Framework • Unit and Integration Testing • Coding Standard • Documentation • Formal Code Review	• Scrum Framework • Unit and Integration Testing • Coding Standard • Documentation • Formal Code Review • Project Estimation and Planning Poker • Use Cases
Environment	• Customer Involvement • Working Environment • Interdependency among Modules	• Customer Involvement • Working Environment • Common Tools and Problems between Teams • Government Projects and the Scrum Method

The concepts in the table represent issues and challenges identified in two firms. The issues and challenges discussed in this section suggest lessons that Scrum practitioners can learn and provide a basis for management guidelines.

STRUCTURED DEVELOPMENT PROCESS

The structured development process consists of systematic process related issues. As shown in the Table 2, some of issues are common to both firms, and these issues reveal the challenges of the Scrum method.

Scrum framework: Most developers at ABC were in favor of the Scrum framework. They thought the Scrum model promoted communication and team work, and helped them keep track of task assignments and monitor task progress. They also thought that working in the Scrum team provided motivation, excitement, and interest. Some developers, however, did not like the frequent daily Scrum meeting. They felt that having the daily Scrum meeting was too much time for not enough value, and that the various Scrum meetings took too much developer's time away from programming, even though the Scrum meetings helped team members refine the goals for each Sprint and improved the quality of products. Some developers and QA personnel also thought monthly Sprint planning meetings took too much time and they want to streamline the planning session. As ABC indicates, any organization should have streamlined Scrum meetings and monitor whether or not various Scrum planning sessions take too much time for not enough value.

The Scrum method works quite well for developers at XYZ and is a big improvement over the waterfall method. Developers thought the Scrum method helped team members get involved in projects, be aware of everyone's progress, contain scope creep, and prevent projects from going too far off course. Like ABC, some developers did not like inefficient Sprint planning and review meetings. They felt that keeping daily Scrum meetings to 15 minutes was difficult because people gab a little too long, there is an excessive amount of material that needed discussed, and taking care of 2 or 3 projects at once. Another problem was setting up the meeting time. It was difficult to get all developers together at one time without interrupting their work because of the flexible work schedule. The research noted that some project managers actually managed the Scrum team rather than let the team self-manage, and that developers did not spend enough time generating a detailed Sprint backlog.

Unit and integration testing: When ABC first adopted the Scrum method, the firm placed a quality assurance (QA) person on each Scrum team. This caused a problem because the QA person is always behind the Sprint schedule. This problem was resolved by creating a QA-only Scrum team that covered all code generated by other Scrum teams. This solution created another problem that when developers made a change and passed the code to the QA team, the QA personnel sometimes did not know the other areas that affected by the changes. To resolve this problem, the firm asked developers to do some portion of the testing themselves that the QA people usually cover. This may not be a good move economically for ABC firm because developers usually are paid more than QA personnel; this might be a sensitive issue that any organization needs to resolve wisely.

XYZ utilized a tool called "N-Unit" for unit testing, and each developer tested his/her own code. Interestingly, the firm also invited their clients to the test site and had them track, test, and enter bugs that they found. However, having developers test their own code had two issues. First, developers usually assume that their code always worked. Second, developers could not as thoroughly test their code as third-party testers.

Another issue was related to large legacy code not designed for unit or integration testing. Developers had a difficult time testing the legacy code. The legacy code and the code working behind the curtain were a big challenge to the firm. The firm also needed to hire more people who had a wide range of testing skills in software. Insufficient client budget also made it difficult to test everything covering one hundred-percent of the code. It seems that the firm utilized the unit test well, but it did not cover all possible combinations of issues due the short client budget and lack of wide range of skilled QA personnel. In addition, the firm needed to rewrite the legacy code or find out an efficient way to make the legacy code unit-testable.

Coding standard: ABC utilized coding standards. They have very specific coding standards in many areas in order to have easily maintainable and expandable code. Most developers agree that having a formal coding standard

enables them to understand other developer's code, but some developers worry about putting too many coding standards on developer's shoulders. Some developers actually think that heavy coercion to the standard may hamper their performance because they have to look at coding documents back and forth to see if their code conforms to the standard.

XYZ did not have a formal coding standard but had a verbal coding standard; developers felt that they were close enough to comment when a person who did not follow the norm. Developers had their own coding style, which was influenced by several commercial software packages, such as Microsoft Visual Studio and Borland. Most developers and project managers thought forcing a coding standard might hamper developer's creativity and hinder performance because they had to relearn how to code in many places.

Documentation: After ABC started the Scrum method, many detail documents, such as class diagrams, sequence diagrams, activity diagrams, communication diagrams, and use cases were significantly reduced, or disappeared. The lack of detailed design caused many problems in complex projects. One main area affected considerably was testing, because QA personnel depend heavily on documentation to find problems. Another problem with a lack of detailed documentation was the tendency to write code without taking time to think about what effects the code may have on other parts of the application. These resulted in an increased number of bugs, which then required a lot of developer working hours to fix. This was a major issue and caused major code re-writes. It's obvious from ABC that if any organizations deal with complex and large projects, they need to tailor the Scrum philosophy on reducing the amount of documentation.

XYZ also reduced the amount of documentations significantly. Developers tried to place more comments and explanations for any tricky logic in the code, along with explanations for any changes that they made, to compensate for the lack of documentation. However, it turned out that many developers had a hard time completing tasks without any documentation, especially developers who needed to work on parts of the system they had never worked on before and new developers who did not have much experience with XYZ's projects. Further, those developers asked a lot of questions, which took much time away from developers who did understand the project. As XYZ indicates, no documents at all are a very dangerous idea that leads to many problems, including causing the agile method to be as slow as anything else.

In the agile methods, the code itself is regarded as all the documentation that developers need. However, it is apparent that zero documents are not always the right way for large-scale and complex projects, especially, in a distributed Scrum environment. The amount of documentation should be decided based on the context of the development environment, though Parnas [24] suggests a wordy document and Simon [28] suggests no more than a two page long document**.**

An additional problem was that only one main developer had extensive knowledge about the firm's systems, rather than every developer on the Scrum team having shared skills and knowledge of the systems. If that main person leaves the firm for any reason, it would be a big problem because it may take several months to recover the knowledge lost. Keeping all team members equal by sharing skills and knowledge on the systems is not easy, and not feasible in reality.

Formal code review: ABC has utilized a web-based formal code review, and developers think the formal code review is a vital and critical process in creating high-quality software applications. Any issues and challenges were not identified at ABC related to the formal code review. Developers at XYZ did not have a formal code review, but they had an occasional informal code review. Not having a formal code review invoked some issues. First, developers did not pay extra attention to their code, because they believed no one would look at it again. If they believed that at some point somebody would go back and look at their code, they would have more accountability. Second, developers lost opportunities to improve the quality of their code and enhance their coding skills through feedback from other developers. Third, there was a high chance that developers wasted time and money by trying to re-invent the wheel from scratch because there are many code examples already written, tested, and proved efficient by the Dot Net framework or other commercial builders. Most developers knew the benefits of having a formal code

review, but they just never had the time to do it. XYZ needs to set a time aside for a formal code review and select an appropriate tool to facilitate the code review.

Project estimation and planning poker: This issue came up only at XYZ, though having an accurate project estimate was an important part of projects for both firms. Developers at XYZ had a hard time estimating the duration of a project, and the level of hardness increased when developers needed to deal with legacy code or when they did not have the experience required to finish a project. However, it seems they mitigated this issue by introducing a new project estimation method called "Planning Poker". A lot of comments from developers revealed that the Planning Poker method provided developers with the opportunity to throw out their honest opinion without being biased or coerced by other developers. Also, Planning Poker helped developers have a chance to discuss estimation gaps between developers, and guided them to reaching better estimates. One noticeable benefit was when developers were able to break big tasks into the smallest measurable segments, they were easily able to make good estimates. The researcher noticed that every developer thought Planning Poker was very useful, effective, and produced reasonable estimates. However, though Planning Poker could help estimation, the bottom line was developers needed to be familiar with the technologies that they were going to use, the business logic, and the system itself. Otherwise, estimation will still be one of the most difficult parts of a project.

Use cases: This issue was identified only at XYZ. Developers at XYZ knew that they could understand the system better with use cases and that they had the best success when they had use cases. Though the firm has reduced the amount of specification documents a lot since the firm adopted Scrum, one Scrum team created a fair amount of use case documentation based on a list of items that team members needed to build. Three issues were identified related to creating use cases. First, some developers were not well prepared to write use cases because they were unfamiliar with a system. Second, clients did not have a clear and precise idea what they really wanted to have in their system. Third, clients did not know what use cases are or how to use them.

The first issue is an in-house issue and the other two are client-related issues. The in-house issue can be resolved through a well-organized employee training program. The client-related issues were resolved by having developers come up with some specifications and having clients review it. It would be better if clients know what they really want to have in their system and understand use cases. If clients have a notion of use cases and have a clear idea of their system, developers can communicate with clients better and create better use cases, which can lead to successful projects.

ENVIRONMENT

Several issues were identified in the environment category. The issues and challenges related to the environmental factor are discussed below.

Customer involvement: Due to the large number of customers scattered across the United States, ABC needed to come up with a different solution to incorporate customer feedback. One way that the firm employed was to send out product line managers to the customer and have them collect project requirements. The project line managers also utilized WebEx to show features of the products and some charts and graphs to reduce the number of onsite visits. Another way the firm employed to get customer feedback was to host a user conference once a year. At the conference, the firm demonstrated new policies and directions of product development. The customers then voted for or against the policies and new development direction. One issue associated with the customer involvement at ABC was that QA people had a difficult time providing the customers with quick bug fixes. This resulted from the Scrum method principles, which required the QA people to focus more on the code generated during the Sprint process than on responding to customer problems.

Though each project at XYZ was only for one customer, developers at XYZ had difficulties getting customers involved in the decision making process. The customers did not willingly participate in the process because they were busy and had other things to do. Poor customer involvement in projects caused problems for the firm because developers needed to create specifics without conversing with clients. Often times it took a lot of hours to figure out what exactly customers really wanted to include in their system, because they did not know what they want in their future system. This was a big roadblock for developers as they went through the development process.

Though XYZ did not have enough customer involvement, most final products were accepted by their customers with minor changes requests. XYZ's case indicates that if developers get together more often with their customers, organizations can deliver a software product to customers sooner with better features and functions. Organizations may also reduce maintenance fees by delivering a more correct product that customers want.

Working environment: There were mixed feelings among developers about the open-space working environment. Some developers liked a cubicle setting because they thought it increased the number of communications between team members and fostered collaboration and the teamwork. However, most developers did not like the open-space working environment because they could not concentrate on their work while their coworkers talked to one another. It was apparent that most developers liked having their own office rather than a cubicle in order to be productive.

Many developers at XYZ liked the open-space working environment because it provided easy access to other developers and it fostered communication. Though some developers enjoyed the open-space working environment, other developers did not like it and thought it brought some downsides and problems. First, developers were easily distracted when their co-worker's talked to other co-workers or when they had a phone conversation with someone. Second, developers were less productive when they could not concentrate because of a lot of background noise. To cancel out the noise, developers utilized headphones, which they put on to drown everything else out. Though this helped most developers, some could not focus on their works just because they have the headphones on.

Interdependency among modules: At ABC, as the size and complexity of the project grew, the dependencies and interconnections among tasks in the application increased. However, developers were not able to fully consider all the dependencies and interconnections among modules because of their narrow-focused planning and design in each Sprint planning meeting. The developers also had a tendency to do things in a quick and dirty way without thinking whether the code would be flexible enough for future needs. As ABC indicates, any organizations should support and encourage developers to spend more time on considering the dependencies and interconnections among modules. The issue of interdependency among modules was not identified at XYZ because XYZ's projects were relatively small and less complex compared to ABC's projects.

Common tools and problems between teams: Though XYZ did not show any signs of interdependencies among modules due to the firm's small size of projects, the firm did have issues with common tools and problems between teams. It appeared that one Scrum team's members could spend many hours finding the right tools or technologies suitable for their project without knowing that other Scrum teams already employed similar tools or technologies. This is a big waste of precious developer's time if two Scrum teams can utilize the same or similar tools or technologies.

The research also noted that each Scrum team had similar problems, which might be resolved using similar solutions. Teams spent time resolving similar problems, each in their own way, which is another waste doing duplicate work if the same solution can be applied to both problems. The firm should appoint a person to inform teams if there are similar technologies that other teams already took advantage of and whether there are similar problems that other teams faced and resolved successfully.

Government project and Scrum method: This issue was identified only at XYZ because the firm has been dealing with many government projects. A government project usually requires heavy documentation, big planning, and big design up front. This does not conform to the philosophy of the Scrum method. It is a big challenge to complete government projects with the Scrum method because the government itself is not agile, and the nature of government is bureaucratic. An additional issue is that developers have to learn the jargon and acronyms used in the descriptions of government projects, adding unproductive time to complete a government project. To overcome these hurdles, XYZ wants to combine the Scrum method with a Unified Process (UP) method in order to conform more to the requirements of government projects. This hybrid of Scrum and the Unified Process has not been developed in the firm.

MANAGEMENT GUIDELINES

This section provides management guidelines to help organizations that are already utilizing Scrum or planning to implement Scrum in the future. The guidelines explained here also help organizations avoid stumbling blocks in their Scrum implementation. The issues and challenges identified and discussed in the previous section provide the basis of the following guidelines.

1. The duration and rules of the daily Scrum meeting should be strictly observed; the duration of other Scrum meetings should be dynamically adjusted based on the agenda for efficiency.
2. Project managers should foster collaboration between developers and QA personnel. Developers should be able to do a unit test of other developer's code and work closely with QA people on integration testing.
3. Organizations should educate developers that every piece of code should be testable and designed for ease of testing.
4. Formal coding standards increase readability and understanding of other developer's code; too many coding standards hamper developer performance.
5. Lack of documentation is a source of problems, especially for large-scale and complex projects. The philosophy of the Scrum method which reduces documentation significantly should be tailored. Organizations need to determine how much documentation is adequate for their projects.
6. Along with documentation, organizations need to promote each Scrum team member having an equal amount of skill and knowledge relative to the project they are working on.
7. Formal code review is a vital and critical process for quality applications. Organizations should establish an efficient way to do formal code reviews.
8. Planning poker is a very easy, useful, and efficient way to evaluate projects. Developers should break a big project into the smallest possible tasks to get better estimates on those tasks.
9. Use cases are important specifications that elevate a developer's understanding of the project that they are working on; both developers and clients need to be educated in how to write good use cases.
10. Customer involvement is very critical for the success of a project. Organizations should invite customers to participate in the decision making process and find out a good way to include them in the various Scrum meetings.
11. If any organizations have a large number of scattered customers, they should consider the use of an annual or semi-annual user conference to explain their new products, collect user feedback, and have them vote for or against the organization's new product direction.
12. Open-space working environments promote teamwork and communication, but organizations should come up with methods to help developers deal with environmental distractions.
13. For large-scale and complex projects, organizations should encourage and support developers spending sufficient time thinking about dependencies and interconnections between modules.
14. If any projects require heavy documentation, big planning, and/or big design up front, the Scrum method might not work well unless combined with another method, such as a Unified Process.

CONCLUSION

This research identified the several critical issues and challenges that may affect the quality of the application of agile methods and illustrated how agile methods can be adopted and utilized to effectively support the development of mission-critical, small- and large-scale projects. This paper also provided management guidelines to help organizations avoid and overcome obstacles in adopting the Scrum method as a future software development method. The lessons about Scrum obtained through the two case studies will be valuable assets to many Scrum practitioners.

REFERENCES

[1] Anacon, D. (1990) Outward bound: Strategies for team survival in an organization, *Academy of Management Journal,* 33, 2, 334-365.

[2] Beck, K. (2000) *Extreme programming explained: Embrace change,* Reading, MA: Addison-Wesley.

[3] Benbasat, I., Goldstein, K. and Mead, M. (1987) The case research strategy in studies of information systems, *MIS Quarterly*, 27, 2, 369-368.

[4] Benbasat, I. and McFarlan, W. (1984) An analysis of research methodologies in the information systems research challenge. (pp. 47-85). Boston, MA: Harvard Business School Press.

[5] Boehm, B. (2002) Get ready for agile methods with care, *IEEE Computer*, 35, 1, 64-69.

[6] Boehm, B. and Turner, R. (2003) Using risk to balance agile and plan-driven methods, *IEEE Computer*, 36, 6, 57-66.

[7] Bonoma, T. V. (1985) Case research in marketing: Opportunities, problems, and a process, *Journal of Marketing Research*, 22, 2, 199-208.

[8] Brooks, F. P. (1995) *The mythical man-month*, Reading, MA: Addison-Wesley.

[9] Charmaz, K. (2002) Qualitative interviewing and grounded theory analysis. In J. F. Gubrium and J. A. Holstein (Eds.), *Handbook of qualitative research*, (2nd ed., pp. 675-694). Thousand Oaks, CA: Sage Publications.

[10] Eisenhardt, K. M. (1989) Building theories from case study research, *Academy of Management Review*, 14, 4, 532-550.

[11] Elsbach, K. D. and Sutton, R. I. (1992) Acquiring organizational legimitacy through illegimate actions: A marriage of institutional and impression management theories, *Academy of Management Journal*, 35, 4, 699-733.

[12] Erickson, F. (1973) What makes school ethnography 'ethnography'?, *Council on Anthropology and Education Newsletter*, 4, 2, 10-19.

[13] Fruhling, A. and Vreede, G. (2006) Field experiences with extreme programming: Developing an emergency response system, *Journal of Management Information Systems*, 22, 4, 39-68.

[14] Gall, D. M., Gall, P. J. and Borg, R. W. (2003) *Educational research: An introduction*, Boston, MA: Allyn and Bacon.

[15] Glaser, B. G. and Strauss, A. L. (1967) *The discovery of grounded theory: Strategies for qualitative reserach*, New York: Aldine Publishing Company.

[16] Glesne, C. (2006) *Becoming qualitative researchers*, Boston, MA: Pearson.

[17] Highsmith, J. and Cockburn, A. (2001, September) Agile software development: The business innovation, *IEEE Computer*, 34, 9, 120-122.

[18] Isabella, L. A. (1990) Evolving interpretations as a change unfolds: How managers construe key organizational events, *Academy of Management Journal*, 33, 1, 7-41.

[19] Kahn, W. A. (1990) Psychological conditions of personal engagement and disengagement at work, *Academy of Management Journal*, 33, 4, 248-266.

[20] Kaplan, R. S. (1985) *The role of empirical research in management accounting*, Boston, MA: Harvard Business School.

[21] Leffingwell, D. (2007) *Scaling software agility: Best practices for large enterprises*, Upper Saddle River, NJ: Addison-Wesley.

[22] Leonard-Barton, D. A. (1990) A dual methodology for case studies: Synergistic use of a longitudinal single site with replicated multiple sites, *Organization Science*, 1, 3, 248-266.

[23] Martin, P. Y. and Turner, B. A. (1986) Grounded theory and organizational research, *The Journal of Applied Behavioral Science*, 22, 2, 141-157.

[24] Parnas, D. (2006) Agile methods and GSD: The wrong solution to an old but real problem, *Communication of the ACM*, 49, 10, 29.

[25] Pettigrew, A. M. (1990) Longitudinal field research on change: Theory and practice, *Organization Science*, 1, 3, 267-292.

[26] Schach, S. R. (2004) *An introduction to object-oriented systems analysis and design with UML and the unified process*, Boston: McGraw Hill.

[27] Schwaber, K. and Beedle, M. (2002) *Agile software development with scrum*, Upper Saddle River, NJ: Prentice Hall.

[28] Simon, M. (2006) Global software development: A hard problem requiring a host of solutions, *Communication of the ACM*, 49, 10, pp. 32-33.

[29] Sommerville, I. (2004) *Software engineering*, Boston: Addison-Wesley.

[30] Stone, E. (1978) *Research methods in organizational behavior*, Glenview, IL: Scott, Foresman and company.

[31] Strauss, A. L. and Corbin, J. M. (1988) *Basics of qualitative research: Techniques and procedures for developing grounded theory*, (2nd ed.)Thousand Oaks, CA: Sage Publications.

[32] Strauss, A. L. and Corbin, J. M. (1990) *Basics of qualitative research: Grounded theory, procedures, and techniques*, Newbury Park, CA: Sage Publications.

[33] Sommerville, I. (2004) *Software Engineering*, Boston, MA: Addision-Wesley.

[34] Sutton, R. I. (1987) The process of organizational death: Disbanding and reconnectiong, *Administrative Science Quarterly*, 32, 4, 542-569.

[35] Turner, B. A. (1983) The use of grounded theory for the qualitative analysis of organizational behavior, *Journal of Management Studies*, 20, 34, 333-348.

[36] Watson, R. T., Kelly, G., Galliers, D. and Brancheau, C. (1997) Key issues in information systems management: An international perspective, *Journal of Management Information Systems*, 13, 4, 91-115.

[37] Williams, L. and Cockburn, A. (2003, June) Agile software development: It's about feedback and change, *IEEE Computer*, 36, 6, 39-43.

[38] Willson, C. D. (2009) A brief introduction to SCRUM: An agile methodology, *Matincor Inc. Information Technology Management Consulting*,

[39] Yin, R. K. (1989a) *Case study research: Design and methods*, Beverly Hills, CA: Sage Publications.

[40] Yin, R. K. (1989b) Research design issues in using the case study method to study management information systems. In J. I. Cash and P. R. Lawrence (Eds.), *The information systems research challenge: Qualitative research methods*, (). Boston, MA: Harvard Business School Press.

[41] Yin, R. K. (1993) *Applications of case study research*, Newbury Park, CA: Sage Publications.