ISSUES AND CHALLENGES OF AGILE SOFTWARE DEVELOPMENT WITH SCRUM

Juyun Cho, Colorado State University-Pueblo, joey.cho@colostate-pueblo.edu

ABSTRACT

Agile software development methods have been developed and evolved since early 1990s. Due to the short development life cycle through an iterative and incremental process, the agile methods have been used widely in business sectors where requirements are relatively unstable. This paper explains the differences between traditional software development methods and agile software development methods, and introduces the characteristics of one of the popular agile methods, Scrum. Finally, the paper illustrates issues and challenges discovered through an in-depth case study in a company which has employed Scrum for many projects. The insights presented in the paper can be used in organizations that are in the process of agile software development using Scrum.

Keywords: Scrum methodology, traditional software development, agile software development, empirical process.

INTRODUCTION

Traditional Software Development Methods (TSDMs) including waterfall and spiral models, utilize extensive planning, codified process, rigorous reuse, heavy documentation and big design up front [2]. Due to these characteristics, TSDMs are often called heavyweight development methods. The TSDMs are still widely used in industry because of their straightforward, methodical, and structured nature [6], as well as their predictability, stability, and high assurance [3].

Though many TSDMs have been developed since the waterfall model to provide significant productivity improvements, none of them are free from major problems including blown budgets, missed schedules, and flawed products [3, 4], and they have failed to provide dramatic improvements in productivity, in reliability, and in simplicity [4]. Due to constant changes in the technology and business environments, it is a challenge for TSDMs to create a complete set of requirements up front.

As a remedy for the shortcomings of TSDMs, a number of Agile Software Development Methods (ASDMs) including Scrum, eXtreme Programming (XP), Crystal, and Adaptive Software Development (ASD), have been developed and evolved since 1990s to embrace, rather than reject, high rates of change [24]. Such new approaches focus on iterative and incremental development, customer collaboration, and frequent delivery [18] through a light and fast development life cycle. Although many positive benefits of agile approaches including shorter development cycle, higher customer satisfaction, lower bug rate, and quicker adaptation to changing business requirements have been reported [3], there have been few empirical field studies on issues and challenges of ASDMs. Therefore, the aim of this research paper was to discover the issues and challenges of one particular agile method in practice, Scrum, through an in-depth case study in a mid-sized, web-based development projects for government.

The remainder of this paper discusses the differences between traditional methods and agile methods, and then presents a brief history, framework, and empirical process of the Scrum methodology. Finally, the paper discusses issues and challenges of the Scrum methodology discovered through an in-depth case study.

TRADITIONAL SOFTWARE DEVELOPMENT METHODS (TSDMs)

One of well-known traditional software development methods is the waterfall model. The waterfall model utilizes a structured progression between defined phases: planning, analysis, design, implementation, and maintenance. The planning phase which occupies typically about 15% of total Systems Development Life Cycle (SDLC) is the fundamental process to identify the scope of the new system, understand why a system should be built, and how the project team will go about building it through technical, economical, and organizational feasibility analysis. The analysis phase, which occupies about 15% of SDLC, analyzes the current system, its problems, and then identifies ways to design the new system through requirements gathering. The design phase (35%) decides how the system will operate in terms of hardware, software, and network infrastructure. The implementation phase (30%) is the actual programming. The maintenance phase (5%) focuses

on go-live, training, installation, support plan, documentation, and debugging [5]. Figure 1 and table 1 below show a typical waterfall lifecycle and deliverables respectively. As we can see in the figure and the table, each phase must be accomplished before the following phase can begin and each phase cannot go back to the previous phase like water in the waterfall cannot climb up once it reaches to lower position.

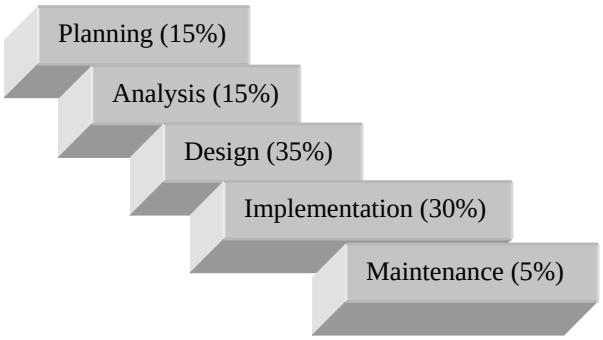

Figure 1 Waterfall model lifecycle

Phases	Deliverables
Planning Phase	Planning Specifications
Analysis Phase	Analysis Specifications
Design Phase	Design Specifications
Implementation Phase	Completed Product

Table 1 Waterfall model deliverables

Over the past four decades, traditional waterfall-style software development methods have been widely used for large-scale projects in the software industry and in the government sector due to their predictability, stability, and high assurance. As mentioned earlier, however, TSDMs have a number of key shortcomings, including slow adaptation to constantly changing business requirements, and a tendency to be over budget and behind schedule with fewer features and functions than specified [2, 6, 16, 19, 21]. Boehm and Phillip [22], and Jones [23] both reported that during their project development experience, requirements often changed by 25% or more. Williams and Cockburn [24] also mentioned that one of problems of TSDMs is the inability to respond to change that often determines the success or failure of a software product.

One interesting research study conducted by the Standish Group of 365 respondents and regarding 8,380 projects representing companies across major industry segments, shows that only a small percentage of projects (16.2%) that used traditional methods were completed on-time and on-budget with all features and functions specified. However, 52.7% of the projects were completed either over-budget, over the time estimate and/or offering less features and functions; 31.1% of projects were canceled at some point during the development cycle [17] (see Figure 2).

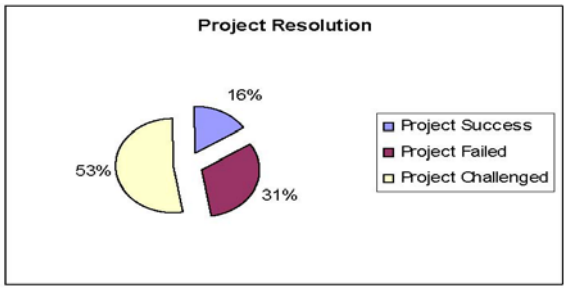

Figure 2 Project resolution (Source: The Standish Group [17])

To overcome these shortcomings, several practitioners developed agile software development methods including Scrum, eXtreme Programming (XP), Crystal, and Adaptive Software Development (ASD). The next section explains the characteristics and principles of agile software development methods.

AGILE SOFTWARE DEVELOPMENT METHODS (ASDMs)

The manifesto for agile software development which was created by seventeen practitioners in 2001 (http://www.agilemanifesto.org), reveals which items are considered valuable by ASDMs. As shown in Table 2, ASDMs concentrate more on 1) individuals and interactions than processes and tools, 2) working software than comprehensive documentation, 3) customer collaboration than contract negotiation, and 4) responding to change than following a plan.

More Valuable Items	Less Valuable Items
Individuals and interactions	Processes and tools
Working software	Comprehesive documentation
Customer collaboration	Contract negotiation
Responding to change	Following a plan

Table 2 Manifesto for agile software development

The twelve principles behind the agile manifesto also present the characteristics of ASDMs (http://www.agilemanifesto.org/principles.html). As

shown in Table 3, ASDMs 1) satisfy the customer through early and continuous delivery of software, 2) embrace changing requirements, even in late development cycle, 3) deliver working software frequently, 4) work daily with business people, 5) facilitate motivated people, provide them with good environment and support, and trust them, 6) assist face-to-face conversation within a development team, 7) use working software as a primary measure of progress, 8) promote sustainable development and keep sponsors, developers, and users moving at a constant pace, 9) pay attention to technical excellence and good design, 10) maintain simplicity, 11) promote self-organizing teams, and 12) foster inspections and adaptations.

#	Principles
1	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2	Welcome changing requiremts, even late in development. Agile processes harness change for the customer's competitive advantage.
3	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4	Business people and devlopers must work together daily throught the project.
5	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6	The most efficient and effective method of conveying informaiton to and within a development team is face-to-face conversation.
7	Working software is the primary measure of progress.
8	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9	Continuous attention to technical excellence and good design enhances agility.
10	Simplicity—the art of maximizing the amount of work not done—is essential.
11	The best architectures, requirements and designs emerge from self-organizing teams.
12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Table 3 Principles behind the agile manifesto

The ASDMs have the potential to provide higher customer satisfaction, lower bug rates, shorter development cycles, and quicker adaptation to rapidly changing business requirements [3, 10, 12].

There are many different characteristics between ASDMs and TSDMs. Boehm [2], for example, reports nine agile and heavyweight discriminators (See Table 4). He believes the primary objective of ASDMs is on rapid value whereas the primary objective of TSDMs is on high assurance. He also believes that ASDMs should be used for small teams and projects. If the size of the team and projects are large he suggests TSDMs.

Project Characteristics	Agile discriminator	Heavyweight Discriminator
Primary objective	Rapid Value	High Assurance
Requirements	Largely emergent, rapid change, unknown	Knowable early, largely stable
Size	Smaller teams and projects	Larger teams and projects
Architecture	Designed for current requirements	Designed for current and foreseeable requirements
Planning and Control	Internalized plans, qualitative control	Documented plans, quantitative control
Customers	Dedicated, knowledgeable, collaborated, collocated onsite customers	As needed customer interactions, focused on contract provisions
Developers	Agile, knowledgeable, collocated, and collaborative	Plan-oriented; adequate skills access to external knowledge
Refactoring	Inexpensive	Expensive
Risks	Unknown risks, Major Impact	Well understood risks, Minor impact

Table 4 Differences between ASDMs and TSDMs (Source: Boehm [2])

SCRUM METHODOLOGY

The Scrum software development process is an agile process that can be used to manage and control complex software and product development using iterative and incremental practices [1] and is an enhancement of iterative and incremental approach to delivering objected-oriented software [13]. The origin of term "Scrum" came from the popular sport Rugby, in which fifteen players on two teams compete against each other. Takeuchi and Nonaka [20] first used rugby strategies to describe hyper-productive development processes in Japan. Three strategies from rugby including a holistic team approach, constant interaction among team members, and unchanging core team members are adopted into Scrum management and control processes.

The Scrum process was developed by Schwaber and Sutherland [15]. The former developed and formalized the Scrum process for system development while he was at his company, Advanced Development methods (ADM), in the early 1990s. The latter developed many of the initial thoughts and practices for Scrum when he was at Easel Corporation as a vice president of Object Technology in 1994. By a joint effort of both Schwaber and Sutherland, the Scrum process was first introduced to public at the conference of Object-Oriented Programming, Systems, Languages and Applications (OOPSLA) in 1996 [13].

Empirical Process Control

The co-founder of the Scrum process, Schwaber argues that the Scrum process employs an empirical process control which has three legs underlying all of its implementations: transparency (visibility), inspection, and adaptation [14, 25]. Transparency or visibility means that any aspects of the process that affect the outcome must be visible and known to everybody involved in the process. Inspection requires that various aspects of the process be inspected frequently enough so that unacceptable variances in the process can be detected. Adaptation requires that the inspector should adjust the process if one or more aspects of the process are in an unacceptable range.

A code review can be analyzed with the empirical process control described above. Any code written by developers should be visible to everybody (transparency). The most experienced and knowledgeable developers can review the code (inspection). If there is a room to improve the code, reviewers' comments and suggestions should be reflected in the code (adaptation).

Framework of Scrum

The framework of Scrum consists of three components including roles, ceremonies, and artifacts [25]. There are three distinct roles in the Scrum process: the Product Owner, the Team and the ScrumMaster. *The Product Owner* is responsible for getting initial and on-going funding for the project by creating the project's overall requirements, return on investment (ROI) objectives, and release plan [25]. *The Team* is responsible for implementing the functionality described in the requirements. Teams should be self-managing, self-organizing, and cross-functional to maximize team performance. All of the team members are responsible for both the success and the failure of sub-systems and of entire systems [25]. The *ScrumMaster (SM)* is responsible for ensuring that Scrum values, practices, and rules are enacted and enforced. The SM represents management and the team to each other [15]. SM also tries to remove any impediments imposed on developers.

There are several ceremonies in the Scrum process including the Daily Scrum Meeting, the Daily Scrum of Scrums Meeting, the Sprint Review Meeting and the Sprint Planning Meeting. *The Daily Scrum Meeting (TDSM)* is a 15-minute status meeting to talk about what has been accomplished since the last meeting, what items will be done before the next meeting, and what obstacles developers have. TDSMs facilitate communications, identify and remove impediments to development, highlight and promote quick decision-making, and improve transparency (visibility) as explained in the previous section. *The Daily Scrum of Scrums Meeting (TDSSM)* is another short daily meeting and follows the same format as a regular TDSM. The main reason for having TDSSM is to synchronize the work between multiple Scrum teams. *The Sprint Planning Meeting (TSPM)* is a monthly meeting, where the Product Owner and Team get together to discuss what will be done for the next Sprint which lasts usually for 30 days. In TSPM, team members break a project into a set of small and manageable tasks so that all the tasks can be completed in one Sprint. *The Sprint Review Meeting (TSRM)* is another monthly meeting which is held at the end of the Sprint. TSRM is usually a four-hour time-boxed meeting, where team members present what was developed during the Sprint to the Product Owner and stakeholders.

In addition to the Scrum roles and ceremonies, the Scrum process provides three artifacts namely the Product Backlog, the Sprint Backlog, and the Burndown Chart. *The Product Backlog* is a collection of functional and non-functional requirements, which are prioritized in order of importance to the business. The items in the Product Backlog are created and maintained by the Product Owner. *The Sprint Backlog* is created by team members from the Product Backlog in a way that the high priority items in the Product Backlog are first selected and broken into a set of smaller tasks. When the Product Backlog items are divided into small tasks, team members estimate the completion time for each task. Team members try to make tasks as small as possible so that every task can be accomplished within three days. The Sprint Backlog consists of these small tasks. *The Burndown Chart* is a graphical presentation where work remaining is tracked on the vertical axis and the time periods tracked on the horizontal axis. The Burndown Chart should be accessible by every member who participates in the project.

Flow of Scrum

The Scrum process begins with a vision of the system and a simple plan on ROI and release milestones. The vision is described in business terms rather than technical terms. The vision may be unclear at first but will become more precise as the project moves forward. As mentioned earlier, the Product Owner is responsible for getting initial funding, delivering the vision while maximizing ROI, and creating the Product Backlog. The prioritized items in the Product Backlog are divided into smaller tasks through the Sprint Planning Meeting and placed in the Sprint Backlog. In the Sprint Planning Meeting, the Product Owner explains the content, purpose, meaning, and intentions of each item in the Product Backlog. Team members can ask questions if they do not understand any items in the Product Backlog. All the tasks in the Sprint Backlog are done through the iteration of the Sprint which consists of the Daily Scrum Meetings. Figure 3 illustrates the flow of the Scrum process.

ISSUES AND CHALLENGES OF SCRUM

Several issues and challenges were discovered through an in-depth case study in a company that has employed Scrum for many small- and medium-size web-based projects. Data were collected through a formal face-to-face interview with nine employees in the company including a vice president of operations, a director of operations, a project manager, a ScrumMaster, and five software engineers. All of the interviews were audio-taped, transcribed, and later coded. In the process of data analysis, grounded theory [7, 8] was used to derive constructs from the immediate raw data. Some of repeated issues and challenges are coded below.

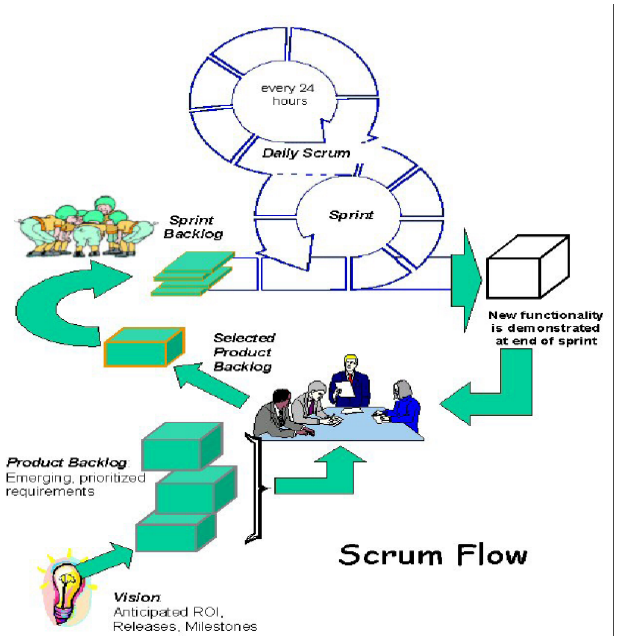

Figure 3 Flow of Scrum (Source: Hodgetts [9])

Documentation

The Scrum method, like other agile software development methods, significantly reduces amount of documentation [13, 14, 15]. In fact, the agile methods claim that the code itself should be a document. That is why developers who are accustomed to agile methods place more comments in the code. Several developers mentioned they placed more explanations for any tricky piece of code and for any changes that they made. However, many developers agree that without having any documents, it is very difficult to complete tasks for those developers who are working on parts of the system they never worked on before and also for new developers who do not have much experience with the project. For both cases, developers who do not understand the project ask a lot of questions, which takes time away from developers who do understand the project. One developer mentioned that "When I first got here, of course, I was overwhelmed. It would have been nice to have some documents that explain why certain things were done in a particular way and what they were." One more developer mentioned that "Agile methods do not use specification documents. I think that might be a weakness in agile methods. The

agile methods allow you to go much quicker as long as whoever is specifying has a very good idea of what clients want. If this is not the case, the agile methods are just as slow as anything else because you are going to have to get clarification."

Another developer also raised the issue of the lack of documentation. He stated "Right now, we have one guy who is the main guy. He knows all of the systems and I think, personally, that might be a mistake. Not because he is not good at it, but because it just makes one gigantic point of failure if he is hit by a bus or if he leaves for another company." It would take several months for the company to recover the knowledge that one main developer has. The idea behind reducing documents in the agile methods is to keep every team members equal by sharing skills and knowledge on the systems. In that way, if one person leaves, there is still a lot of shared knowledge that has gone around among other team members, so it is not a big deal. However, in reality, this is not feasible.

Communication

It is well known that ineffective communication is the root of most failures in software products [11]. The Scrum process recognizes the important role of communications in the software development process and provides an excellent means of communication. All interviewees agree that the Daily Scrum Meetings improve communications between team members within a team. However, each team in the company is fairly separate and generally there is not much communication between teams. The lack of communication between teams could cause problems such as duplicated work. This problem can be solved or at least mitigated if the company holds the Daily Scrum of Scrums Meeting because SMs from each Scrum team can make sure no work is being duplicated.

Good within-team and between-team communication can be accomplished through the framework of Scrum, but communication with the customer can be problematic. Several developers mentioned that "the biggest area of communication issues that we have is with the customer more than anything else because they tend to not give us a lot of feedback." Part of the reason that the customer does not provide feedback is that, in most cases, they have other daily jobs to take care of in addition to the work with developers. This is related to the customer involvement issue which is explained in the next section.

Customer Involvement

Customer involvement in the software development process is very critical to the success of the project. The agile methods state that the customer should be part of the development process from analysis and design to implementation and maintenance. However, the case study reveals that developers have difficulties working with customers on the projects. A project manager mentioned that "Customers are not involved in the decision making process until it is all done." He also stated that "We don't get as much customer involvement as we want. Our customers are busy and they have other things to do than to talk to programmers all day." One developer complained that "We request our customers to talk to us every day and at least minimum once a week but they are not very involved so we end up with talking with maybe twice per Sprint." Another developer stated that "Our customers did not give us specification documents. We basically had an hour-long meeting to make a specification. So, it was vague when we started it. It was up to us to make specifics and estimations. I think the biggest roadblock in our development process was in the customer involvement. Though we did not have enough customer involvement, our customers accepted most part of the system that we created and asked us for minor changes. But I think it would be much better if we get together more often with our customers."

It seems that, most of the time, customers do not know what they really want in their future system and it becomes a roadblock for customers to get involved in the project development process. One developer stated that "Customers think they have a clear idea but they do not. For example, the customer wants to track people's credit. To them, that's clear and precise. But to us, we need to know who the people are, what the credits are, when they expire, how long we track them, what rewards earned for many credits." Due to unclear customer requirements, developers have a hard time figuring out what exactly the customer wants to include in their system. One ScrumMaster mentioned "We need to get out a lot of information from unclear statement, which takes more time, which causes us to get involved less because it takes too much time. But we don't have any other way to do it because we don't have information."

Working Environment

Most agile methods including Scrum recommend removing the cubicles and setting up collocated team space because cubicles promote isolation and the Scrum process relies heavily on high-bandwidth,

face-to-face communication, and network [25]. The open space is considered better than the cubicles and private offices in the Scrum process. Many developers like the idea of open-space-working environment. One developer mentioned "I feel like I am little closer to other developers in open space. It's really nice to be able to look across the room and talk to somebody else in the team and ask questions quickly. I don't feel like I am shouting over the cubicle wall to get to them." Another developer stated that "Open space is good because everyone is easily accessible. I like it because I think it fosters communication. It's very easy to say hey, I need some help, information, or come, look at this. Everyone is just kind of opening, and it seems to work very well."

Though some developers enjoy the open-space-working environment, other developers do not like the open space and they mentioned downsides and some problems. One developer stated that "the open areas are very nice to communication but it does hurt when you try to concentrate because there are a lot of distractions. For example, when co-workers are having a conversation with somebody or having a phone conversation, it's very distracting." Another developer mentioned that "I am less productive because a lot of noises are going all around. Without having cubicle walls or private offices, the distractions are pretty high which is hard to work with." A team lead stated that "You know the best working environment is an office. In your private office, you can do things your way, and focus on things without being distracted by other noises."

To cancel out the noises, most developers use headphones. The director of operations stated that "Everybody has headphones and they can just put those on and listen to something. That pretty much drowns everything else out. However, several developers complain that "We, developers, are usually working while listening to music. We all have a nice headphones workout. Everything is going under that. But if I need to focus on something, that's really difficult just because I have headphones on."

Scrum Ceremonies

Scrum ceremonies including the Daily Scrum Meeting, the Sprint Planning Meeting, and the Sprint Review Meeting, seemed to help software engineers develop high-quality systems. Most developers testified that the Scrum ceremonies have been very useful and very productive. Several developers mentioned that "the 15-minute standup Daily Scrum meeting has allowed us to be on the same page because we can talk to each other and everybody knows what everybody else is working on.

However, some developers talked about inefficient Sprint Planning and Review Meetings. One developer argued that "Some of our Sprint Meetings are so simple and it seems to be a waste of time spending a whole day just for planning and review. I think it needs to be adjusted based on the complexity of the project that we are working on." Another developer mentioned that "Our daily standup Scrum meetings sometimes go on a little longer just because everybody is talking about what they did last night. I think there probably are some good advices on trying to keep your daily standup meetings consistent and short so that people are not distracted and they can go back to work quickly as most people would rather work productively than waste a time." Another issue is related to setting up the meeting time. Due to the flexible work schedule among developers, it is difficult to get together all at one time. One developer stated that "I think the hard things for us in Scrum is when to do it because some of us get in at 7:30 am and some of us at 9:30 am. So as a team, we just have Scrum as soon as everyone gets in. That's usually at ten or eleven. The problem is that those who get in early are interrupted from their work because they've been working for two or three hours very well. They are in the group or zone so being interrupted and it's frustrating. We talked about doing it at the end of the day but that also has a problem because some people come in at 6:30 am and leave at 3:30 pm, and some people come in at 9:30 am and leave at 6:30 pm. It makes hard for our team to get together all at one time."

CONCLUSIONS

Agile software development methods were developed to provide more customer satisfaction, to shorten the development life cycle, to reduce the bug rates, and to accommodate changing business requirements during the development process. This paper presents characteristics of traditional software development methods and agile software development methods, and the differences between them. This paper also introduces the roles, ceremonies, and artifacts of Scrum, which is one of the most well-known agile software development methods in the industry. This paper also presents five issues and challenges including documentation, communication, user involvement, working environment, and Scrum ceremonies, discovered through an in-depth case study in a software company that makes small- and mid-size web-based applications. If the five issues and challenges are addressed and resolved before the

project is launched, organizations will have fewer difficulties in producing high-quality software products using Scrum.

The author wants to thank Dr. Sherry Marx at Utah State University for her insightful advice on a qualitative research method.

REFERENCES

1. Advanced Development Methods, Inc. (2007). Scrum, Retrieved March 19, 2008, from http://www.chaos.com.
2. Boehm, B. (2002, January). Get ready for agile methods with care. *IEEE Computer, 35*(1), 64-69.
3. Boehm, B. & Turner, R. (2003, June) Using risk to balance agile and plan-driven methods. *IEEE Computer, 36*(6), 57-66.
4. Brooks, F. P. (1995). *The mythical man-month.* Reading, MA: Addison-Wesley.
5. Dennis, A., Wixom, B. H., & Tegarden, D. (2005). *Systems analysis and design with UML version 2.0*. Hoboken, NJ: Wiley.
6. Fruhling, A. & Vreede, G. (2006). Field experiences with extreme programming: Developing an emergency response system. Journal of Management Information Systems, 22(4), 39-68.
7. Gall, D. M., Gall, P. J., & Borg, R. W. (2003). *Educational research: An introduction*. Boston, MA: Allyn and Bacon.
8. Glesne, C. (2006). *Becoming qualitative researchers: An introduction*. Boston, MA: Allyn and Bacon.
9. Hodgetts, P. (2005). Product development with Scrum. Retrieved March 1, 2008, from http://www.agilelogic.com.
10. Miller, K., & Larson, D. (2005, winter). Agile software development: Human values and culture. *Technology and Society Magazine, IEEE, 24*(4), 36-42.
11. Parnas, D. (2006). Agile methods and GSD: The wrong solution to an old but real problem. *Communication of the ACM, 49*(10), 29.
12. Parrish, A., Smith, R., Hale, D., & Hale, J. (2004). A field study of developer pairs: Productivity impacts and implications. *IEEE Software, 21*(5), 76-79.
13. Schwaber, K. (1996). SCRUM development process. *Proceedings of ACM SIGPLAN on Objected-Oriented Programming, Systems, Languages, & Applications (OOPSLA '96),* San Jose, California.
14. Schwaber, K. (2007). What is Scrum? Retrieved March 5, 2008, from http://www.scrumalliance.org/system/resource/file/275/whatIsScrum.pdf.
15. Schwaber, K., & Beedle, M. (2002). Agile software development with Scrum, Upper Saddle River, NJ: Prentice Hall.
16. Schach, S. R. (2004). *An introduction to object-oriented systems analysis and design with UML and the unified process.* Boston: McGraw-Hill.
17. Standish Group (1994). The chaos report. Retrieved March 6, 2008, from http://www.standishgroup.com/sample_research/chaos_1994_1.php.
18. Stazinger, J. W., Jackson, R. B., & Burd, S. D. (2005). *Object-oriented analysis & design with unified process.* Boston: Thomson Course-Technology.
19. Summerville, I. (2004). *Software Engineering.* Boston: Addison-Wesley.
20. Takeuchi, H., & Nonaka, I. (1986, January-February). The new new product development game. *Harvard Business Review*, p137-146.
21. Watson, R. T., Kelly, G., Galliers, D., & Brancheau, C. (1997). Key issues in information systems management: An international perspective. *Journal of Management Information Systems, 13*(4), 91-115.
22. Boehm, B. & Papaccio, P. (1988). Understanding and controlling software costs. *IEEE Transactions on Software Engineering, 14*(10), 1462-1477.
23. Jones, C. (1997). *Applied software measurements: Assuring productivity and quality.* McGraw Hill.
24. Williams, L. & Cockburn, A. (2003, June). Agile software development: It's about feedback and change. *IEEE Computer, 36*(6), 39-43.
25. Schwaber, K. (2004). *Agile project management with Scrum.* Redmond, WA: Microsoft Press.