Agile Project Management

Warren W. Tignor

SAIC

472 Cornwall Court

Severna Park, MD 21146 USA

410-647-9652

wtignor@ieee.org

International Conference of the System Dynamics Society

Albuquerque, NM 2009

July 26 – July 30, 2009

This paper explores agile project management relative to the conceptual models documented by Lyneis & Ford (2007) to examine whether the generic structures and the findings for those structures apply, or new ones are required.

Table of Contents

List of Tables

List of Figures

1 Introduction

The International System Dynamics Society has supported the art and science of project management for decades. Several domains of project management researched using system dynamics include hardware development projects, software development projects, defense projects and civil construction projects. Lyneis & Ford (2007) published that one of the most successful areas for the application of system dynamics has been project management in terms of new system dynamics theory, new and improved model structures, number of applications, number of practitioners, value of consulting revenues, and value to clients.

This paper explores agile project management relative to the conceptual models documented by Lyneis & Ford (2007) to examine whether the generic structures and the findings for those structures apply to agile project management.

2 Statement of the Problem

The questions addressed by this paper are whether agile project management has a unique structure or will fit within the generic conceptually formed system dynamic project management structures identified by Lyneis & Ford (2007). Questions to be addressed include the following:

1. Does agile project management use feedback responses?
2. Are agile project management tasks causally linked?
3. What are agile project management typical behaviors?
4. What are the limits of agile project management, if any?
5. What are some lessons learned to date and future research directions regarding agile project management and system dynamics?

3 Literature Review

The literature review consisted of reviewing agile project management publications. The articles reviewed met the criteria for analysis by containing some information regarding project management, agile development, feedback, tasking, behavior, and limits. These were indicators that the article would be relevant to the eventual step to compare their content to the touchstone article by Lyneis & Ford (2007), essentially the baseline for system dynamics and project management.

The first article is an interesting introduction to agile methodology and project management. According to Ramsin & Paige (2008, p. 7), most agile methodologies incorporate explicit processes, keeping them as lightweight as possible. There usually is an iterative-incremental process present. Of particular interest to this paper is the Scrum agile methodology for software development as described by Schwaber & Beedle (2001), Schwaber (2004), & Schwaber (2007).

Ramsin & Paige (2008, p. 58) provide a diagram of the Scrum pattern from research by Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002), see Figure 1.

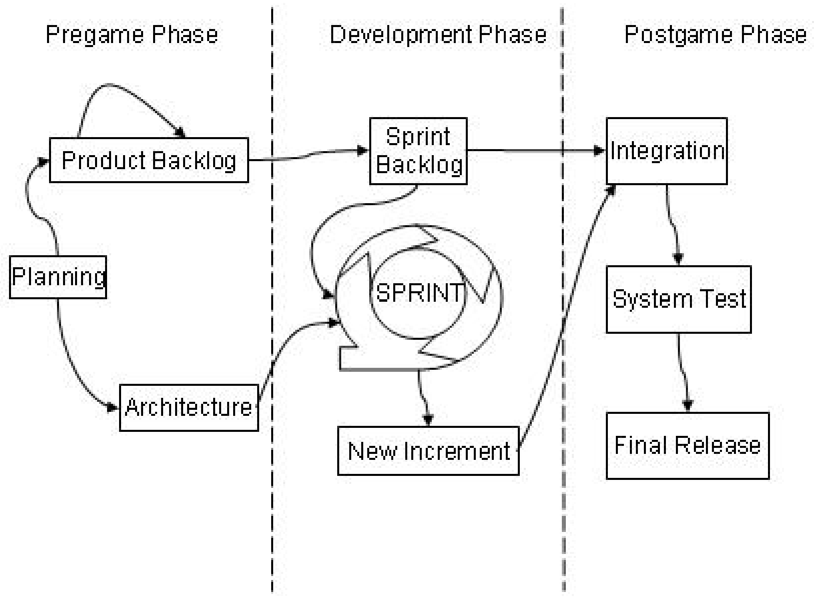

Figure 1 Scrum Pattern (Adapted from: Ramsin & Paige, 2008, p. 58)

The Scrum process consists of three phases as follows:

1. Pregame is concerned with setting the stage for the iterative-incremental development effort and consists of the following subphases[1]:

1.1 Planning is focused on producing an initial set of prioritized requirements, i.e., listing the Product backlog, analyzing risks associated with the project, estimating the resources needed for implementing the requirements, obtaining the resources necessary for starting the development, and determining an overall schedule.

1.2 Architecture/high-level design focuses on determining the overall architecture of the system in such a way as to accommodate the realization of the requirements in Sprint and Product backlog.

2. Development focuses on iterative and incremental development of the system. Each iteration (Sprint) is typically one month in duration and delivers an operational increment satisfying a predetermined subset of the Product backlog[2].

3. Post-game focuses on integrating the increments produced and releasing the system into the user environment[3].

At the end of each Sprint, according to Ramsin & Paige (2008, p. 60), the increment produced is demonstrated to all the parties concerned. A comprehensive assessment is made of the achievements of the Sprint in satisfying the Sprint goal, and the Product backlog is updated accordingly. Fully realized requirements are marked as such, necessary bug fixes or enhancements[4] are added, and appropriate changes are made to partially developed requirements. The Sprint can also result in the identification of new requirements, or changes to already defined requirements, for consideration when updating the product backlog[5]. Another objective of the Sprint review meeting is to discuss and resolve issues impeding the progress of the development team. The meeting also addresses updating the system architecture according to the insights gained during the sprint.

For their embedded control system project, Cordeiro, L., Mar, C., Valentin, E., Cruz, F., Patrick, D., Barreto, R., & Lucena, V. (2008) adapted agile principles and patterns, focusing on issues related to system constraints and safety. They required strong unit testing to ensure timeliness and correctness. They needed a platform-based design approach to balance cost and time-to-market, regarding performance and functionality constraints. They found that the agile methodology significantly reduced the design time and cost as well as resulted in better software modularity and reliability.

Cordeiro et al. (2008) developed a methodology named as TXM (The neXt Methodology) composed of practices from Software Engineering, Scrum and eXtreme Programming (XP).

Their methodology contained three different process groups that were used during the system development project: system platform, product development and product management. The system platform processes were the following: product requirements, system platform, product line, and system optimization. The product development processes were as follow: functionality implementation, task integration, system refactoring, and system optimization. The product management processes consisted of the following processes: product requirements, project management, bug tracking, sprint requirements, product line, and implementation priority. The TXM process life cycle consisted of five phases: Exploration[6], Planning[7], Development[8], Release[9] ,[10] and Maintenance[11,12].

The experiment with the TXM methodology as reported by Cordeiro et al. (2008) successfully balanced cost and time-to-market in view of performance and functionality constraints. Additionally, they concluded that the project methodology led to better software modularity and reliability.

Denning, Gunderson & Hayes-Roth (2008, p. 31) said, referring to the waterfall development model, "There is too much at stake to continue to allow us to be locked into a process that does not work." They asserted that development time is the critical factor where the user environment changes often and significantly in as little as 18 months (Moore's Law), Denning et al., 2008, p. 289. In government and large organizations, the bureaucratic acquisition process for large systems can often take a decade or more, which contributes to delivered systems being unsatisfactory for the customer. They are in favor of the evolutionary project process as opposed to the waterfall.[13] To them, the astonishing success of evolutionary development projects challenges common sense. They believe that the evolutionary development project approach may be the only way to achieve satisfactory replacements for aging large systems and to create new, unprecedented systems.

To substantiate their position, Denning et al. (2008, p. 30) cite the U.S. government's 2004 World Wide Consortium for the Grid (W2COG) experiment which took advantage of a provision of acquisition regulations that allows Limited Technology Experiments (LTEs).

The experiment developed a secure service-oriented architecture system, comparing an LTE using evolutionary methods against a standard acquisition process. In 18 months, the LTE's process delivered a prototype open architecture that addressed 80 percent of the government requirements, at a cost of $100,000. In contrast, after 18 months at a cost $1.5 million, the standard process delivered only a concept document that neither provided a functional architecture, nor had a working prototype.

Jiang & Eberlein (2008, p. 10) address the "methodology war" issue between agile and non-agile (e.g., waterfall) projects by comparing their similarities and differences using the ***CHAPL*** framework: (1) ***C***ontextual analysis, (2) ***H***istorical analysis, (3) ***A***nalysis by analogy, (4) ***P***henomenological analysis, and (5) ***L***inguistic analysis. The framework inputs are best practices of classical systems engineering (SE) methodologies, agile methodologies, and industry practices[14].

Although the CHAPL framework was in its early stage of research, the authors believe that it will help to understand the relationship between the methodologies, and support rational selection of best practices and suitable SE methodologies for a software project.

They recognized the fact that there is no model that allows us to reason about the suitability of SE methodologies in any particular circumstance, and advocate developing a reasoning mechanism that assists in SE methodology selection. This "reasoning model" is part of their larger research vision and the CHAPL framework its basis.

Tarr, Williams, & Hailpern (2008) examined software development project processes and software development organizations as adaptive and emergent entities. Their research looks at software development processes and organization as having properties that are unknown *a priori*, and resulting from ongoing and continuous response to externals, e.g., evolving requirements, new enterprise priorities, and changes in resources.

They cite the relative success of "bottom-up" agile software processes and the common failure of "top-down" designed software processes, which has led to the postulate that software development has more in common with complex adaptive systems than with assembly lines. They identified that it is necessary for both the project processes and organizations to adapt to ever-changing requirements and evolutionary pressures. They posited that only emergent processes, and adaptive organizations were suited to software development.

Particularly interesting questions they posed are as follows (Tarr et al., 2008, p.22): (1) "What are the feedback loops and mechanisms that are required to ensure an adequate stakeholder/development organization awareness, adaptation, and co-evolution? (2) What is the frequency of feedback required to achieve effective coevolution?"

They presented four feedback loops involving the stakeholders of a software development project and the development team[15]. Figure 2 illustrates postulated feedback between these two populations.

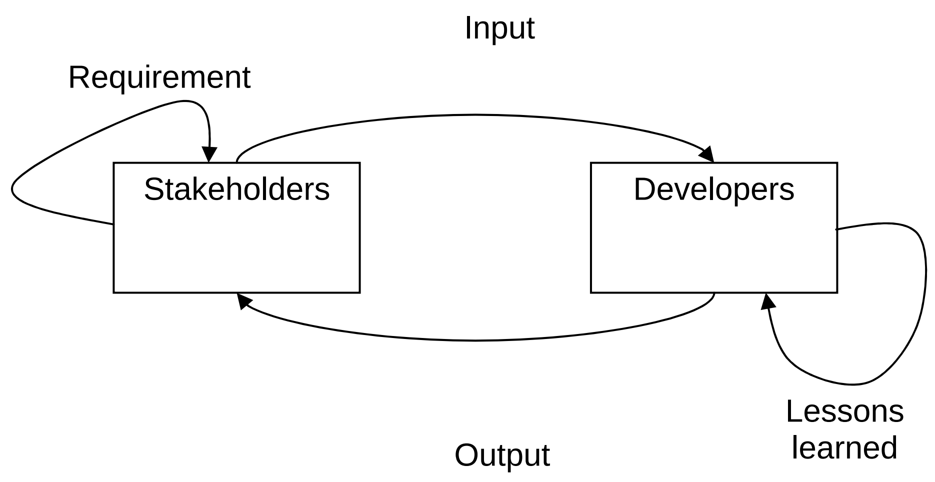

Figure 2 Adapative Development Feedback Loops (Adapted from: Tarr et al., 2008, p.23)

Tarr et al. (2008) acknowledge that they have seen some feedback and control occurring in agile project processes, especially ones involving Scrum. But, they have also seen feedback in top-down project processes in the form of evaluations and acceptance tests from customers.

They note that there is little in the way of understanding these feedback loops. They proposed additional research was needed to understand the feedback loops, how they interact, how they are affected by various pressures (both internal and external), and how each one can be exploited to maximize the probability that a good result will emerge[16]. For example they cited Schwaber & Beedle (2001) that all attempts at external control during a Scrum "Sprint" result in reduced efficiency of the development team in achieving their primary goals. Scrum shields a sprinting team from external control input to whatever extent possible, designating specific control points before and after a Sprint.

Bass (2006) noted that agile approaches have issues when organizations begin to distribute work geographically. The nature of the developing organization changes and practices that worked with collocated groups may no longer work with distributed groups. Two strategies to deal with distributed agile development discussed were: decoupling the work and augmenting the lack of face-to-face communication.

The problem they identify with these strategies is that project management has no means of monitoring the effectiveness of such practices. Most current project management approaches do not explicitly recognize the role of "cognitive synchronization," e.g., mental models, in software development[17]. When it comes to monitoring and controlling projects, managers typically use documents or artifact-driven approaches.

Bass (2006) said that agile project approaches are appealing because managers get to see concrete results regularly. He claimed that agile practices are focused on optimizing the extent to which the team shares a common mental model. Although this has been working well inside of a single team, there is no means, currently, for synchronizing across teams.

There are several ways in which managers might measure either directly or indirectly the extent to which a shared mental model exists (Bass, 2006, p.36): monitor coordination through social network analysis, administer a lightweight Web-based survey (typically less than 10 minutes per individual) on a bi-weekly basis. Additionally, managers could monitor the social networks of the project, for example: (1) Who in the project does an individual coordinate with and how often? (2) How aware is an individual in the activities, skills, and roles of others in the project? (3) To what extent and about what are teams coordinating? (4) Who is involved with particular kinds of decisions in the project?

Traditionally at Microsoft, each product group determined and followed its own practices and processes. Most projects followed a modified waterfall model with daily or weekly integrated-builds and six- to twelve-week design-implement-stabilize cycles (Brechner, 2005, p.40). Unit tests, code reviews, and design inspections varied widely across groups.

The agile software movement gained a great deal of momentum at Microsoft. The focus was on the customer, responding to change, and delivering value instead of artifacts[18]. Developers targeted their rapid development on satisfying customer needs with constant feedback and an uninterrupted value stream.

They found that when serving a wide variety of customers and working with a large number of partners, it was difficult to get them all in a room once a year, let alone once a week or month. They saw this problem as inherent with integrated services and the agile methodologies.

Microsoft's approach was to add "just enough" project process and documentation to keep trust boundaries synchronized with the value stream (Brechner, 2005). This enabled teams to have the latest builds available quickly so that they could validate and adjust direction frequently with customers and partners. Many teams employed Scrum, lean development, test-driven development, and refactoring to be more responsive to change. This helped the teams to produce better designs, minimize work in progress, and provide the required level of code quality.

"Just enough" documentation meant integrated systems had good points of integration. Since an interface change could be disruptive, extra controls were put in place to minimize the possible impact. Interfaces were designed upfront. Versioning of interfaces was as important as versioning of source code and binaries. Versions were well-managed and maintained to keep the entire enterprise up and running without interruption or failure.

"Just enough" was more than a "handshake" according to Brechner (2005, p.41). It meant documenting requirements and interactions of interfaces with buyoff from all key customers and partners. This happened on two levels: between groups of people and between groups of services and included a fallback position to keep the system stable and diffuse tensions.

Detweiler (2007) wrote that agile projects operated under highly compressed time scales and tended to lack traditional project-management processes and skills, relying heavily on team self-governance. The compressed time scales made it difficult to get access to the right customers at the right times. Contrary to popular recommendations, close customer contact and regular feedback may have occurred only sporadically. The agile phases they followed are below[19]:

Phase 1 - Understand users. Detweiler believes a key challenge in contrast to plan-based approaches is that few, if any, background documents or specifications are available to help provide context for requirements. He recommended lobbying hard to have dedicated Sprints/milestones allocated for gathering requirements, especially at the beginning of an agile project to involve developers in synthesizing the data gathered from customer visits.

Phase 2 – The challenge was documentation of requirements. Detweiler (2007) recommended promoting agile-friendly use-case methods so that time and effort were not wasted. By explicitly writing down users' goals, the steps needed to achieve them, and the data needed to support the steps, it would be possible to accelerate the production of prototypes and working code (Detweiler, 2007, p.43).

Phase 3 - Designing the user interface was considered a challenge because independently empowered teams evolved code in parallel, without coordinating with each other. He showed prototypes to design partners and target end-users to gather their feedback early and often to avoid inconsistencies.

Ferreira & Cohen (2008) developed and tested a research model that hypothesized the effects of five characteristics of agile project systems development (iteration, continuous integration, test-driven design, feedback, and collective ownership) on two dependent stakeholder satisfaction measures: (1) stakeholder satisfaction with the development process and (2) stakeholder satisfaction with the development outcome. They focused on Scrum as the agile methodology practice. They intended to better understand general characteristics of agile methodology that lead to improved project outcomes. They recognized that organizations were still learning to blend or balance the characteristics of agile methodology practice, and were still trying different combinations of options.

They defined the Scrum agile characteristics and developed their hypotheses based upon them. There was a hypothesis for each agile methodology characteristic and one for the linkage of process satisfaction and outcome satisfaction. The research results supported each hypothesis (Ferreira & Cohen, 2008, p.53):

1. Iterative development hypothesis: The more iterative the systems development process, the greater the stakeholder satisfaction. Iterative development had a strong effect on stakeholder satisfaction allowing for reprioritization of features and early and continuous demonstrations of system value[20].

2. Continuous integration hypothesis: The more continuous integration was present within the systems development process, the greater the stakeholder satisfaction. Continuous integration supported the early detection of errors to keep the development project on track[21].

3. Collective ownership hypothesis: The greater the degree of collective ownership during the systems development process, the greater the stakeholder satisfaction. Collective responsibility, role swapping, and empowerment of all team members to own the software code benefitted the project process[22].

4. Test-driven design hypothesis: The more test-driven the systems development process, the greater the stakeholder satisfaction. Test-driven design appeared to help focus developers on the delivery of working code that benefited stakeholders[23].

5. Feedback hypothesis: The greater the degree of customer feedback within the systems development process, the greater the stakeholder satisfaction. Regular feedback helped organizations recognize necessary requirements changes by allowing customers ample time to voice their desired changes, which in turn, allowed customers to get what they wanted. The feedback characteristic showed the least variation and the highest mean. Feedback appeared the most commonly applied agile practice, suggesting a move toward user-centered design[24].

6. Process and outcome hypothesis: The greater the stakeholder satisfaction with the systems development process, the greater the stakeholder satisfaction with the systems development outcome. The more satisfied stakeholders were with the development process, the more satisfied they were with the overall project outcome[25].

Nerur, Mahapatra, & Mangalaraj (2005) noted that agile methodologies required letting go of command-and-control management to leadership-and-collaboration[26]. They said that agile methodologies dealt with unpredictability by relying on people and their creativity rather than on processes like traditional methodologies. The project manager's traditional role of planner and controller had to be altered to that of a facilitator[27]. The project manager's role in agile projects became that of coordinator of collaborative development efforts. The project manager's goal was to ensure that the creative ideas of all participants were reflected in the final decision. To this end, the project manager had to relinquish authority enjoyed by traditional methodologies.

To Nerur, Mahapatra, & Mangalaraj (2005, p.77), the principles of agile methodologies paralleled the ideas delineated in Checkland's Soft Systems Methodology and Ackoff 's Interactive Planning. Therefore, they concluded that agile methodologies reflected the essential characteristics of complex adaptive systems.

Augustine, Payne, Sencindiver & Woodcock (2005) reported that traditional project management approaches were based on linear development methodologies and mismatched to today's dynamic systems that must be able to change at "Internet speed"[28]. To their experience, project managers fell back on traditional linear approaches, even when they use agile methodologies.

They examined projects that employed agile methodologies as complex adaptive systems. They crafted a complex adaptive system Agile Project Management (APM) framework to manage agile development projects. The APM framework they prescribed consists of the following practices:

1. Organic teams of from seven to nine members that are self-organizing and emergent. The agile manager establishes clear roles and responsibilities to ensure proper team alignment and accountability.

2. Guiding vision to help anticipate and adapt to changing conditions. A project vision as a simple statement of project purpose will have a powerful effect on individual member behavior.

3. Simple rules that result in complex behavior that emerges over time. The manager identifies practices to provide simple generative rules without restricting the autonomy and creativity of team.

4. Free and open access to information. Information about plans, progress, objectives, and organization is a catalyst for adaptation by each member of the project team.

5. Light-touch management style that replaces traditional control approaches that fail when neat linear tasks don't easily accommodate dynamic processes and schedules require frequent updating to reflect changing circumstances.

6. Adaptive leadership that balances on the edge of chaos. Systems with too much structure are too rigid, while systems without enough structure spiral into chaos. Nonlinear behavior can be positive or negative in a project context and result in unintended outcomes. Adaptive leadership employs "systems thinking" to understand all project forces. System archetypes are used to help identify the unintended and counterintuitive consequences of actions when cause and effect are not closely related in time and space.

Abrahamsson, P., Warsta, J., Siponen, M., & Ronkainen, J. (2003) performed a comparative analysis of agile method's life-cycle coverage, project management support, type of practical guidance, fitness-for-use and empirical evidence. Their results showed that agile software development methods covered certain but different development phases and that most of them did not offer adequate support for project management. Yet, many methods strived for universal solutions as opposed to situation appropriate.

They studied various agile methodologies, e.g., adaptive software development (ASD), agile modeling (AM), extreme programming (XP), Internet-speed development (ISD), and the Scrum approach (Scrum).

In general, Abrahamsson et al. (2003) stated that each method should be efficient, concerning time and resource. They saw that efficiency required project management activities to enable the execution of software development tasks. They saw project management as a support function that provided the backbone for efficient software development, concluding that project management was a relevant dimension in the evaluation of agile software development methods.

A sample comparative analysis of results by Abrahamsson et al. (2003) is shown in Table 1. Each agile method is divided into bars. The top bar indicates whether a method supported project management. The bottom bar shows whether a method relied mainly on abstract principles (white color) or provided concrete guidance (gray color). The length of the three bars shows the phases of the software development cycle supported by the agile method.

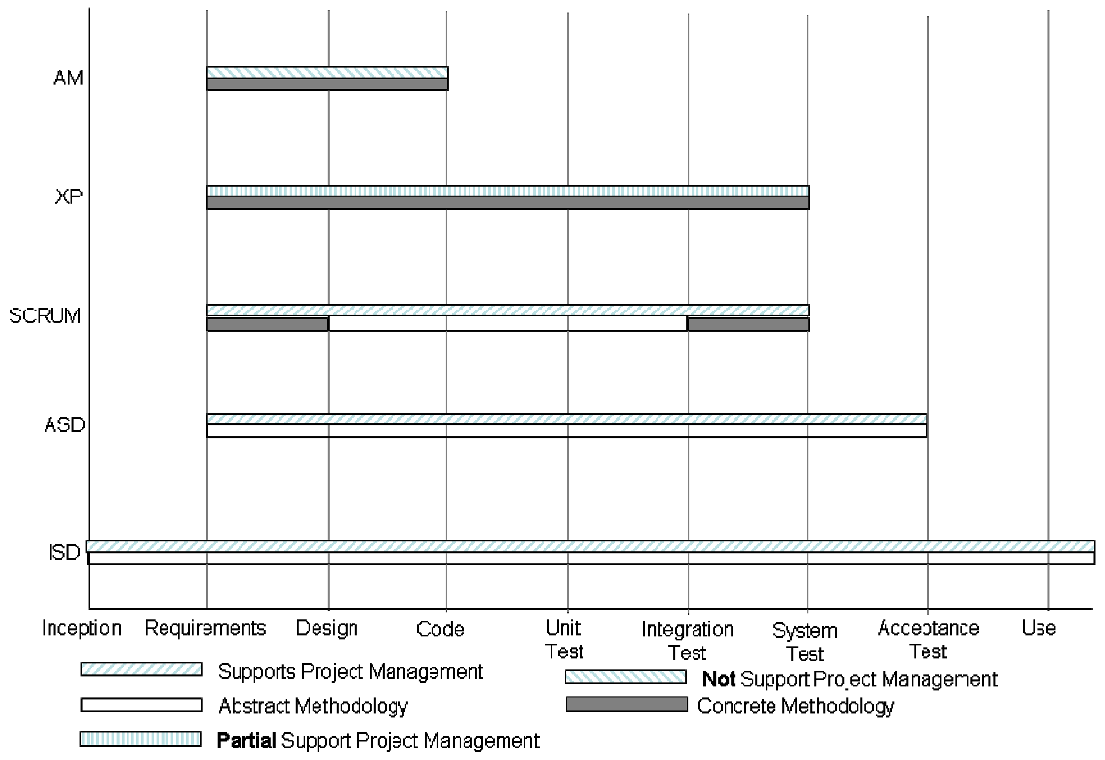

Table 1 Comparison of agile life-cycle, project management, and concrete guidance (Adapted from: Abrahamsson et al., 2003, p. 248)

They concluded that agile software development methods had a wide range of project management coverage[29]. For example, AM did not address project management. XP did not offer a comprehensive project management view. Scrum explicitly addressed managing agile software development projects from requirements specifications through integration test.

They were clear that project management could not be neglected. They say that true project management support was scarce. From a method feasibility point of view, efficient project management was of utmost importance when agile principles such as daily builds, and short release cycles were followed. Additionally, since release and daily builds differed from one method to another, this invited more confusion than clarity. They maintained that project management considerations needed to be addressed explicitly to ensure the alignment between developers and project manager.

Vanderburg (2005, p.543) examined feedback in agile processes[30]. Regarding Scrum, he described its overall structure as a series of iterations, where each iteration is a central feedback mechanism. Although Scrum's iterations typically take 30-day Sprints, after each Sprint, there was a Sprint Review designed to understand the Sprint's success or failure and whether adjustments were needed for the next Sprint. Within each Sprint, the core feedback mechanism was the daily Scrum meeting[31]. The fact that feedback was gathered in a whole-team Scrum meeting probably amplified its effect.

Vanderburg (2005) cited the creators of Scrum, Schwaber & Beedle (2002), that Scrum employs the empirical process control model. Scrum regularly inspects activities to see what was occurring and empirically adapted activities to produce desired and predictable outcomes.

Ko, DeLine, & Venolia (2007) hypothesized that during software projects, little is known about what information software developers look for and why they look for it, e.g., What information is needed to triage bugs? What and why is knowledge sought from their coworkers? What is looked for when they search source code or use a debugger? They thought that by identifying the types of information that software developers seek, they would better understand what tools, processes and practices would help.

To understand developer information needs in more detail, they performed a two-month field study of software developers at Microsoft, focusing on three specific questions:

1. What information do software developers seek?
2. Where do developers seek this information?
3. What prevents them from finding information?

They observed several developer information needs and situations, e.g., the most difficult to satisfy being design questions regarding the intent behind existing code[32] and code yet to be written[33]; nearly impossible to find were bug reproduction steps and the root causes of failures[34]; and lastly, general information seeking was deferred because the coworkers who had the knowledge were unavailable[35].

Ko, DeLine, & Venolia (2007) said that one approach to reducing this communication burden was to automate the acquisition of information. Another approach to address these information needs was through project process change, for example agile methods. The frequent need to consult coworkers for information was an important motivation for Scrum meetings and radical collocation. For example, Chong and Siino (2006) compared interruptions among radically collocated pair programmers versus cubicle-base solo programmers and found that the agile team's interruptions were shorter, more on-topic, and less disruptive.

For an industrial Web-conferencing system, Graham, Kazman, & Walmsley (2007) described their experiences with making project architectural tradeoffs between performance, availability, security, and usability, in light of stringent cost and time-to-market constraints. Traditional requirements elicitation techniques, e.g., questionnaires, surveys, observation, and focus groups, proved to be of limited value in their Web-conferencing domain. They stated that in many cases, Web-conferencing systems were unprecedented, and end users and integrators did not know what they wanted. They took note of Augustine, Payne, Sencindiver & Woodcock (2005) that even minor changes can produce unanticipated effects[36]. Even as systems become more complex and their components more interdependent, the market will not accept failure to adapt to changing requirements and market conditions.

This tension between minor changes and complexity created the classic "agility versus discipline" problem (Graham et al., 2007). They wanted to provide new capabilities quickly, and respond to customer needs rapidly, while designing for long-term extensibility and modifiability.

Given that there were too many unknowns and too many uncontrollable factors, e.g., third-party hardware and software in a multitude of versions, they compensated for the difficulty in analyzing architectural tradeoffs by adopting an agile architecture project discipline combined with a rigorous program of experiments aimed at answering tradeoff questions. The experiments proved invaluable in resolving tradeoffs by helping to turn unknown architectural parameters into constants or ranges of values.

Graham et al., (2007) said that the primary lessons learned were following:

1. It was enormously difficult to anticipate required changes to a system's architecture during the initial design phase,

2. The benefits of using an incremental, agile approach to change were significant to the project's success.

3. While they could have initially architected the system for scalability and fault tolerance, there were not strong requirements at the beginning, and addressing them would have significantly increased time-to-market and development expense.

4 Research Method and Design

The research method consisted of using Lyneis & Ford's (2007) project management system dynamic exposition and structure represented as a stock and flow diagram as a basis for comparison to the agile project descriptions in the literature reviewed.

The research method was designed to test the hypothesis that an agile project management conceptual system dynamics model (stock and flow structure, Figure 3) could be identified in the literature reviewed or whether another structure may be required. In essence, the question was whether the Lyneis & Ford (2007) system dynamic project model holds for agile projects. In particular, the research method was designed to see whether the literature reviewed would fit in any of the four system dynamic structures below as defined by Lyneis & Ford, 2007:

1. A rework cycle: The rework cycle is a canonical system dynamics structure that drives much of the dynamics of project management models.

2. Project control: Controlling systems is the objective of applying system dynamics in many domains, in order to deliver on time, on budget, and with the quality and specifications required. Modeling the controlling feedback loops to close gaps between project performance and targets directly applies system dynamics to project management.

3. Ripple effects: "Ripple effects" is the name commonly used to describe the primary concept of policy resistance to well-intended project control efforts.

4. Knock-on effects: "Knock-on effects" commonly describe the secondary impacts of project control behavior that results in adverse, unintended consequence, e.g., excessive or detrimental negative morale.

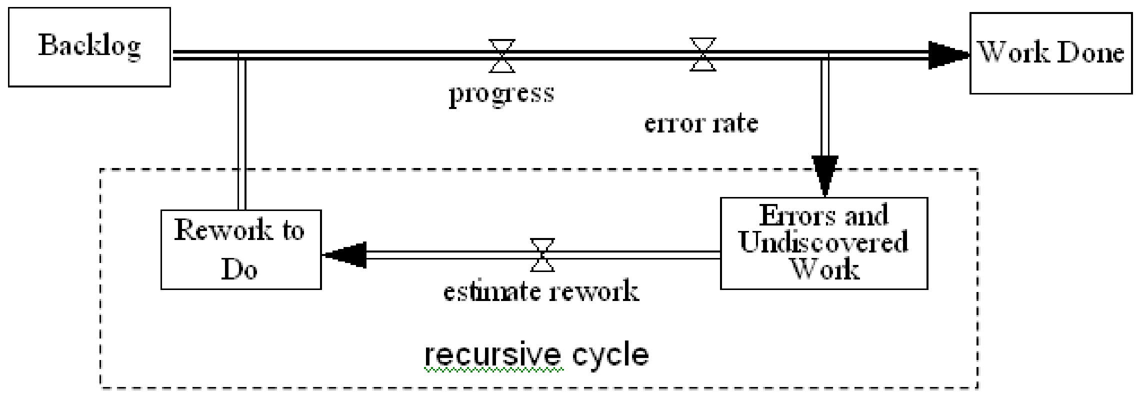

Figure 3 Conceptual Project Management Model (Adapted from: Lyneis & Ford, 2007, p. 161)

The data collection method consisted of reading the review literature and endnoting it relative to the system dynamic structure. Where possible, the method was to endnote the literature to the stocks, rectangle accumulators, and flows, triangle valves. If the review literature fit multiple parts of the system dynamic structure, it was endnoted as such. In instances where the review literature identified work that did not fit any of the four system dynamic categories, it was endnoted as “nonspecific.”

As further clarification of the Lyneis & Ford (2007) standard for comparison to the review literature, it is important to note that they considered the rework flow as the most important single feature of system dynamics project models. Because of the rework cycle’s recursive nature, it has the potential to generate more rework that generates more rework, etc., creating problematic project management behaviors that result in schedule and resource management impacts.

5 Data Analysis

The endnotes made during the review of the literature were collected as comma separated variable (.csv) files for analysis using Microsoft Corporation’s Office Excel®. The overall results are available in Table 2, sorted by Stock and Flow subset, i.e., the stock and flow categories as used by Lyneis & Ford (2007). The overall data show that all but two references, i.e., Denning et al. (2008) and Jiang & Eberlein, (2008) could be partitioned to the model.

Some of the reviewed literature could be allocated to one or more specific parts of the project management stock and flow chart. A summary of the subset results follows in Table 3:

Endnote #	Stock & Flow Subset	Reference	
16	Controlling Feedbacks	(Tarr et al., 2008)	1
24	Controlling Feedbacks	(Ferreira & Cohen, 2008)	2
30	Controlling Feedbacks	(Vanderburg, 2005)	3
10	Error Generation	(Cordeiro et al., 2008)	1
13	general support	(Denning et al., 2008)	1
14	general support	(Jiang & Eberlein, 2008)	2
17	Knock-on Effects	(Bass, 2006)	1
27	Knock-on Effects	(Nerur et al., 2005)	2
33	Original Work to Do	(Ko et al., 2007)	1
1	Original Work to Do	(Ramsin & Paige, 2008)	2
6	Original Work to Do	(Cordeiro et al., 2008)	3
7	Original Work to Do	(Cordeiro et al., 2008)	4
19	Overall	(Detweiler, 2007)	1
35	Overall	(Ko et al., 2007)	2
15	Overall	(Tarr et al., 2008)	3
18	Overall	(Brechner, 2005)	4
20	Overall	(Ferreira & Cohen, 2008)	5
21	Overall	(Ferreira & Cohen, 2008)	6
22	Overall	(Ferreira & Cohen, 2008)	7
26	Overall	(Nerur et al., 2005)	8
28	Overall	(Augustine et al., 2005)	9
29	Overall	(Abrahamsson et al., 2003)	10
31	Overall	(Vanderburg, 2005)	11
2	Progress	(Ramsin & Paige, 2008)	1
8	Progress	(Cordeiro et al., 2008)	2
34	Rework Discovery	(Ko et al., 2007)	1
23	Rework Discovery	(Ferreira & Cohen, 2008)	2
25	Rework Discovery	(Ferreira & Cohen, 2008)	3
5	Rework to Do	(Ramsin & Paige, 2008)	1
12	Rework to Do	(Cordeiro et al., 2008)	2
36	Ripple effects	(Graham et al., 2007)	1
4	Undiscovered Work	(Ramsin & Paige, 2008)	1
11	Undiscovered Work	(Cordeiro et al., 2008)	2
32	Work Done	(Ko et al., 2007)	1
3	Work Done	(Ramsin & Paige, 2008)	2
9	Work Done	(Cordeiro et al., 2008)	3

Table 2 Overall data sorted by Stock and Flow subset

Stock & Flow Subset	Total
Controlling Feedbacks	3
Error Generation	1
Knock-on Effects	2
Original Work to Do	4
Overall	11
Progress	2
Rework Discovery	3
Rework to Do	2
Ripple effects	1
Undiscovered Work	2
Work Done	3

Table 3 Summary of reference count in Stock and Flow subsets

6 Major Findings and Significance

The major finding of this research confirms the hypothesis that elements of agile project management are present in the system dynamics project management model. The aggregate of the reviewed literature contained elements that could be identified with the Lyneis & Ford (2007) model. This finding has significant meaning because a new project management structure, from a system dynamics perspective, is not needed to describe and analyze agile projects. This finding further substantiates that the work of Lyneis & Ford (2007) has identified an archetypical project management pattern.

Regarding whether the literature reviewed would fit in any of the system dynamic structures defined by Lyneis & Ford, 2007, i.e., rework cycle, project control, ripple effects, and knock-on effects, the data confirm that they are met in part, but not as a whole, by any of the reviewed agile project articles. This is significant because the agile community recognizes the presence of these structures but has overlooked the synergy of their relationships. This is important because of the possible impact to agile project management system dynamics could have, e.g., the possibility of recognizing the recursive nature of error generation, undiscovered work, rework discovery, and rework path as illustrated in Figure 3.

6.1 Does agile project management use feedback responses?

Tarr et al. (2008) presented four feedback loops involving the stakeholders of a software development project and the development team as illustrated in Figure 2. This is significant as an initial view of the role feedback plays in an agile project management process.

Ferreira & Cohen (2008) indicated that the greater the degree of customer feedback within the systems development process, the greater the stakeholder satisfaction. Regular feedback helped organizations recognize necessary requirements changes by allowing customers ample time to voice their desired changes, which in turn, allows customers to get what they wanted. The significance is the recognition that feedback is central to a move toward user-centered design.

Vanderburg (2005, p.543) regarded Scrum as an overall feedback structure with a series of iterations. He described Scrum feedback, as gathered in a whole-team setting, to probably have an amplifying effect. The recognition of the possibility that feedback could have an amplifying effect is significant from a system dynamics perspective as a positive feedback loop.

6.2 Are agile project management tasks causally linked?

Ko,et al. (2007) observed several developer information needs that appeared causally linked, e.g., design questions regarding the intent behind existing code and code yet to be written; bug reproduction steps and the root causes of failures; and lastly, general information seeking deferred because the coworkers who had the knowledge were unavailable. Seeing causality is a significant step toward creating system dynamic models to study and manage effects.

Ferreira & Cohen (2008) developed a hypothesized linkage between agile methodology characteristics and stakeholders. They studied the effects of five characteristics of agile systems development, i.e., iteration, continuous integration, test-driven design, feedback, and collective ownership, on two dependent stakeholder satisfaction measures: (1) stakeholder satisfaction with the development process, and (2) with the development outcome. They intended to better understand general characteristics of agile methodology that lead to improved project outcomes and in the process have identified a significant role for system dynamics to address.

Augustine, et al. (2005) report that traditional project management approaches were based on linear development methodologies and mismatched to today's dynamic systems that must be able to change at "Internet speed." To their experience, project managers fell back on traditional linear approaches, even when they used agile methodologies. They sought a linkage between adaptive leadership employing "systems thinking" to understand all project management forces. They believe that system archetypes were a useful link to help identify the unintended and counterintuitive consequences of actions when cause and effect are not closely related in time and space. This is significant because of the recognition that there are unintended consequences and policy resistance when relapsing to traditional management techniques. Linear management policy will not prevail at Internet speeds.

Cordeiro, et al. (2008) adapted agile principles and patterns in order to build embedded control systems focusing on issues related to system constraints and safety. They found strong linkage between unit testing to ensure timeliness and correctness and agile project tasks. They needed a platform-based design approach to balance costs and time-to-market regarding performance and functionality constraints. There is a significant role for system dynamic models supporting the tradeoffs needed in agile development projects.

6.3 What are agile project management typical behaviors?

Augustine, et al. (2005) crafted a complex adaptive system Agile Project Management (APM) framework to manage agile development project behavior. The APM framework they constructed consists of the following practices:

1. Organic teams of from seven to nine members that are self-organizing and emergent.
2. Guiding vision to help anticipate and adapt to changing conditions.
3. Simple rules that result in complex behavior that emerges over time.
4. Free and open access to information.
5. Light-touch management style that replaces traditional control approaches.
6. Adaptive leadership to balance on the edge of chaos.

There is a significant role for system dynamics to help define and clarify the "simple rules" as they apply to "complex behavior." Additionally, system dynamics is a key tool to help anticipate and adapt to changing conditions.

According to Ramsin & Paige (2008, p. 7), most agile methodologies incorporate explicit processes, keeping them as lightweight as possible. There usually is some iterative-incremental process present. Three typical behaviors that they noted were as follows: (1) Pregame - concerned with setting the stage for the iterative-incremental development effort, (2) Development (game) - focused on iterative and incremental development of the system, and (3) Post-game - focused on integrating the increments produced and releasing the system into the user environment. System dynamics is an appropriate tool to study the parameters of these behaviors, their sensitivities and their dynamic interactions. Otherwise, the project manager will have to guess at what effect changing one variable will have on another without any empirical support, a situation that has been proven to generally exceed our capacity to dynamically manage.

6.4 What are the limits of agile project management, if any?

Abrahamsson, et al. (2003) performed a comparative analysis of agile methods' life-cycle coverage, project management support, type of practical guidance, fitness-for-use, and empirical evidence. Their results showed that agile software development methods cover certain but different development phases and that most of them do not offer adequate support for project management. Yet, many methods strived for universal solutions as opposed to situation-appropriate use. There is a significant opportunity for system dynamics to provide a holistic approach to a very dynamic agile development projects.

Bass (2006) says that agile approaches have issues when organizations begin to distribute work geographically. The nature of the developing organization changes and practices that worked with collocated groups may no longer work with distributed groups. Two strategies to deal with distributed agile development are to decouple the work or augment the lack of face-to-face communication. System dynamics could help identify and manage the limits of positive and negative feedback loops that will be present during distributed agile development.

Brechner (2005) found agile software movement gained a great deal of momentum at Microsoft due to its focus on the customer, responding to change, and delivering value instead of artifacts. Developers targeted their rapid development on satisfying customer needs with constant feedback and an uninterrupted value stream.

Unfortunately, a limit was reached when serving a wide variety of customers and working with a large number of partners. Brechner (2005) reports that it was difficult to have customer contact once a year, let alone once a week or month. Microsoft's approach was to add "just enough" process and documentation to keep trust boundaries synchronized and the value stream flowing (Brechner, 2005). A significant question is the limit of "just enough" process as a function of number of developers, requirements, and customer meetings. Answering these questions is a role for system dynamics.

6.5 What are some lessons learned to date and future research directions regarding agile project management and system dynamics?

Graham, et al. (2007) described their experience making architectural tradeoffs among performance, availability, security, and usability, in light of stringent cost and time-to-market constraints in an industrial Web-conferencing system. Their primary lessons learned were as follows:

1. It is enormously difficulty to anticipate required changes to a system's architecture during the initial design phase.

2. The benefits of using an incremental, agile approach to change were significant to the project's success.

For future research, Graham, et al. (2007) are strong proponents of experimentation to compensate for the difficulty in analyzing architectural tradeoffs, given many unknowns and uncontrollable factors. Significantly, system dynamics is an appropriate tool to model and simulate the lessons learned and support their tradeoff experimentation.

Jiang & Eberlein (2008) started researching a framework to compare best practices of classical SE methodologies, agile methodologies, and industry practice. They recognized the fact that there is no model that allows us to reason about the suitability of SE methodologies in any particular circumstance, and advocate developing a reasoning mechanism that assists in agile methodology selection. Such a model is part of their larger research vision and the CHAPL framework their basis for comparison. With a CHAPL framework, system dynamics might make a contribution to the reasoning model necessary to assist in agile methodology selection.

Tarr, et al. (2008) have learned to view software development processes and software development organizations as adaptive and emergent entities. Their research looks at software development processes and organization as having properties that are unknown *a priori*. The properties result from ongoing and continuous response to externals, e.g., evolving requirements, new enterprise priorities, and changes in resources. Particularly interesting questions they pose for future research are as follows: (1) What are the feedback loops required to ensure adequate stakeholder/development organization awareness? and (2) What is the frequency of feedback required to achieve effective coevolution? System dynamics is a significant feedback analysis tool to support the research advocated by Tarr, et al. (2008).

7 Conclusions

In conclusion, the work of Lyneis & Ford (2007) stands the test of agile project management. The literature reviewed shows that there are many areas of agile project management that could benefit from knowledge about and application of the Lynies & Ford (2007) model. Overall, the literature shows that there is little applied science attention to the agile project management process. The holistic approach encapsulated by Lyneis & Ford (2007) could benefit the agile project management process. Of paramount interest is a way to monitor and control the potential impact of recursive error detection and rework to the agile development process.

8 References

Abrahamsson, P., Salo,O., Ronkainen, J., & Warsta, J. (2002). Agile Software Development Methods: Review and Analysis. VTT Publications, Oulu, Finland.

Abrahamsson, P., Warsta, J., Siponen, M., & Ronkainen, J. (2003). New Directions on Agile Methods: A Comparative Analysis. IEEE.

Augustine, S., Payne, B., Sencindiver, F. & Woodcock, S. (2005). Agile Project Management: Steering from the Edges, Communications of the Association of Computing Machinery. 48:(12), pp.85-89.

Bass, M. (2006). Monitoring GSD Projects via Shared Mental Models: A Suggested Approach. GSD'06, Shanghai, China.

Brechner, E. (2005). Journey of Enlightenment: The Evolution of Development at Microsoft. ICSE'05, St. Louis, MO.

Chong, J., Siino, R. (2006). Interruptions on Software Teams: A Comparison of Paired and Solo Programmers. Computer Supported Cooperative Work. Banff, Alberta. pp.28–39.

Cordeiro, L., Mar, C., Valentin, E., Cruz, F., Patrick, D., Barreto, R., & Lucena, V. (2008). An Agile Development Methodology Applied to Embedded Control Software under Stringent Hardware Constraints. ACM SIGSOFT Software Engineering Notes January 2008 Vol. 33 No. 1.

Denning, P., Gunderson, C., & Hayes-Roth, R. (2008). The Profession of IT Evolutionary System Development. Communications of the Association of Computing Machinery. 51:(12), pp.29-31.

Detweiler, M. (2007). Managing UCD Within Agile Projects. ACM Interactions, XIV.3, May/June, pp.40-42.

Ferreira, C. & Cohen, J. (2008). Agile systems development and stakeholder satisfaction: a South African empirical study. In *Proceedings of the 2008 Annual Research Conference of the South African institute of Computer Scientists and information Technologists on IT Research in Developing Countries: Riding the Wave of Technology* (Wilderness, South Africa, October 06 - 08, 2008). SAICSIT '08, vol. 338. ACM, New York, NY, 48-55.

Graham, Nicholas. C.T., Kazman, R. & Walmsley, C. (2007). Agility and Experimentation: Practical Techniques for Resolving Architectural Tradeoffs. 29th International Conference on Software Engineering (ICSE'07).

Jiang, L, & Eberlein, A. (2008). Towards A Framework for Understanding the Relationships between Classical Software Engineering and Agile Methodologies. APSO'08, Leipzig, Germany.

Ko, J. A., DeLine, R., & Venolia, G. (2007). Information Needs in Collocated Software Development Teams. 29th International Conference on Software Engineering (ICSE'07).

Lyneis, J., Ford, D. (2007). System dynamics applied to project management: a survey, assessment, and directions for future research. <u>System Dynamics Review, Vol.23.No. 2/3</u>, pp. 157-189.

Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of Migrating to Agile Methodologies. <u>Communications of the ACM Vol. 48, No. 5</u>.

Ramsin, R. & Paige, R. F. (2008, Feb). Process-Centered Review of Object Oriented Software Development Methodologies. <u>ACM Computing Surveys, Vol. 40, No. 1, Article 3</u>.

Schwaber, K. (2004). <u>Agile Project Management with Scrum</u>. Microsoft Press, Redmond, WA.

Schwaber, K. (2007). <u>Enterprise Scrum</u>. Microsoft Press, Redmond, WA.

Schwaber, K. & Beedle, M. (2001). <u>Agile Software Development with Scrum</u>. Prentice-Hall, Englewood Cliffs, NJ.

Tarr, P., Williams, C., & Hailpern, B. (2008). Toward Governance of Emergent Processes and Adaptive Organizations. <u>SDG'08</u>, Leipzig, Germany.

Vanderburg, G., (2005). A Simple Model of Agile Software Processes– or –Extreme Programming Annealed. <u>OOPSLA'05</u>. San Diego, California.

9 Endnotes

[1] Rework cycle; Original Work to Do; (Ramsin & Paige, 2008)
[2] Rework cycle; Progress; (Ramsin & Paige, 2008)
[3] Rework cycle; Work Done; (Ramsin & Paige, 2008)
[4] Rework cycle; Undiscovered Work; (Ramsin & Paige, 2008)
[5] Rework cycle; Rework to Do; (Ramsin & Paige, 2008)
[6] Rework cycle; Original Work to Do; (Cordeiro et al., 2008)
[7] Rework cycle; Original Work to Do; (Cordeiro et al., 2008)
[8] Rework cycle; Progress; (Cordeiro et al., 2008)
[9] Rework cycle; Work Done; (Cordeiro et al., 2008)
[10] Rework cycle; Error Generation; (Cordeiro et al., 2008)
[11] Rework cycle; Undiscovered Work; (Cordeiro et al., 2008)
[12] Rework cycle; Rework to Do; (Cordeiro et al., 2008)
[13] Nonspecific; general support; (Denning et al., 2008)
[14] Nonspecific; general support; (Jiang & Eberlein, 2008)
[15] Rework cycle; Overall; (Tarr et al., 2008)
[16] Rework cycle; Controlling Feedbacks; (Tarr et al., 2008)
[17] Rework cycle; Knock-on Effects; (Bass, 2006)
[18] Rework cycle; Overall; (Brechner, 2005)
[19] Rework cycle; Overall; (Detweiler, 2007)
[20] Rework cycle; Overall; (Ferreira & Cohen, 2008)
[21] Rework cycle; Overall; (Ferreira & Cohen, 2008)
[22] Rework cycle; Overall; (Ferreira & Cohen, 2008)
[23] Rework cycle; Rework Discovery; (Ferreira & Cohen, 2008)
[24] Rework cycle; Controlling Feedbacks; (Ferreira & Cohen, 2008)
[25] Rework cycle; Rework Discovery; (Ferreira & Cohen, 2008)
[26] Rework cycle; Overall; (Nerur et al., 2005)
[27] Rework cycle; Knock-on Effects; (Nerur et al., 2005)
[28] Rework cycle; Overall; (Augustine et al., 2005)
[29] Rework cycle; Overall; (Abrahamsson et al., 2003)
[30] Rework cycle; Controlling Feedbacks; (Vanderburg, 2005)
[31] Rework cycle; Overall; (Vanderburg, 2005)
[32] Rework cycle; Work Done; (Ko et al., 2007)
[33] Rework cycle; Original Work to Do; (Ko et al., 2007)
[34] Rework cycle; Rework Discovery; (Ko et al., 2007)
[35] Rework cycle; Overall; (Ko et al., 2007)
[36] Rework cycle; Ripple effects; (Graham et al., 2007)