An Evolutionary Lifecycle Model with Agile Practices for Software Development at ABB

Aldo Dagnino
ABB US Corporate Research Center
1021 Main Campus Drive
Raleigh, NC, USA
aldo.dagnino@us.abb.com

Abstract

Current software environments face increased pressure to develop products under evolving requirements, changing technologies, scarce human resources, and the need to develop high quality applications. ABB has similar pressures, especially as the organization embraces the Industrial IT initiative that aims at the development of interoperable and intelligent products. A new generation of software development lifecycle models has emerged lately called "Agile" and they embrace change, reduce development cycle time, and attempt a useful compromise between no process and too much process. The Agile Development in Evolutionary Prototyping Technique (ADEPT) presented in this paper, was developed at the ABB US Corporate Research Center, and incorporates "Agile" practices to streamline the technology development lifecycle of Industrial IT products.

1. Introduction

Agile software development focuses on the talents and skills of individuals in software development teams and molds processes to specific people and teams [4]. The most obvious difference between plan-driven development lifecycle models and agile development lifecycle models is that agile models are less document-oriented and place more emphasis on code development. Agile models are more adaptive than predictive, and they are more people-oriented than process-oriented. Self-organization and intense collaboration within and across organizational boundaries characterize agile organizations. Agility requires that teams have a common objective, mutual trust and respect, a collaborative but speedy decision-making process, and the ability to positively deal with ambiguity. Agile models provide two important attributes that allow managing the current complexities of new technologies, continuous emergence of requirements, and a potentially unstable workforce: (a) **emergence** and (b) **self-organization**.

As a software application emerges, the development organization can take advantage of early releases to capture market share, improve its competitive advantage, and gather feedback from real users. Agile development teams can respond to continuous emerging of requirements and unstable technologies by self-organizing and adapting themselves to the work at hand. Instead of being restrained by roles and quickly obsolete task plans, teams can adapt to the work with the available resources [3, 5, 7, 8, 10]. Agile processes involve teams working together to determine how to convert an emergent list of requirements into a software product through an iterative process. Agile models employ an empirical process control mechanism to manage and control projects. Teams self-organize and determine the best way to convert requirements and technology into product increments. Management uses frequent inspections to control the process. Task and work definitions are few and far between in agile processes. Teams are agile because they are not constrained by "heavier" descriptions of "what to do". Teams are provided with adequate and sufficient guidance for working but are not told details on how to do their job [11].

2. The Planning Continuum

If all the processes of a software development lifecycle can be fully defined so that they can be designed in a repeatable fashion with predictable results, they are known as *defined processes*. If all aspects of the processes of a lifecycle model are not fully known then they are called

Proceedings of the Eighth IEEE international Conference on Engineering of Complex Computer Systems (ICECCS'02)
1050-4729/02 $17.00 © 2002 IEEE

empirical processes. Empirical processes require close watching and control, with frequent intervention. They are not repeatable and require constant measurement and control through intelligent monitoring [11].

Both, defined processes and empirical processes, and therefore, plan-driven and "Agile" models are best suited to be used in different situations. Figure 1 shows the planning continuum that development models can provide a software manager when developing software applications. This planning continuum is an adaptation of Boehm's work [2]. The definition of planning refers to the amount of documentation, schedules, specification of roles and responsibilities, work breakdowns, procedures, reporting, etc. On the far left of the continuum, there is the totally undisciplined software development or "hacking". Moving towards the right of the continuum, Extreme programming (XP) [1, 9] and Scrum represent agile development lifecycles that compared with unplanned and undisciplined hacking emphasize a good amount of planning. Moving further to the right of the continuum of Figure 1 plan-driven methodologies offer tools for planning, documenting and keeping tight control of projects. At the end of the continuum there is the micro-management of projects where excessive pre-determined plans tend to over-constrain the development team.

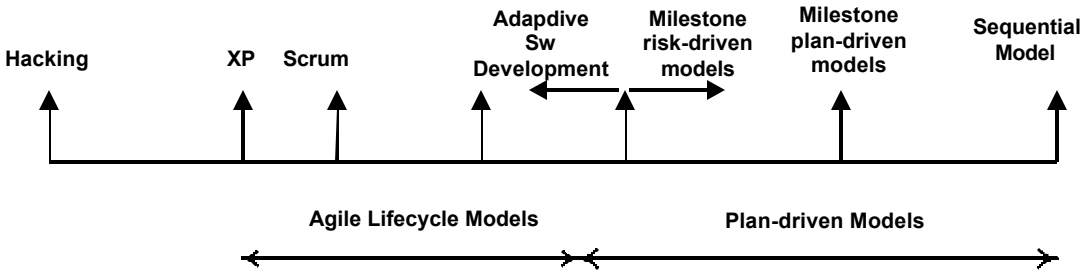

Figure 1. The Planning Continuum (adapted from Boehm, 2002, p. 65)

3. Agile Lifecycle Models for Software Development

Agile models employ an iterative and incremental delivery approach as a response to changing and emergent requirements. Self-organizing teams perform the work and have very little definition on how to perform the work. Few artifacts and documents are explicitly required. Agile lifecycle development models use frequent inspections for managing and controlling the development work, in an adaptive fashion. Typically, lifecycle models are not completely agile or completely defined. Some agile models use defined methods to manage risk such as empirical inspections after development iterations that increase agility. An example of a completely agile methodology is "hacking", but the risk of this approach is unacceptable, and the resultant software systems are usually not maintainable. The agile lifecycle models reviewed in this section include (a) Extreme Programming (XP); (b) Scrum; (c) Dynamic Systems Development Method (DSDM); and (d) Feature Driven Development (FDD). These agile models are full frameworks for development, and they can be used "out-of-the-box" to build software systems. DSDM and FDD require modeling and requirements definition phases prior to beginning iterations. Scrum and XP use iterations that generate code from the very beginning. Table 1 presents a list of various characteristics that make lifecycle models more agile [11].

Table 1. Characteristics of Agile Development Lifecycle Models (adapted from [11])

No	Characteristic	Description
1	***Process***	***Flexibility and ease of use***
1a	*Scalability*	Ability to support any size of project
1b	*Minimal planning to define projects*	Minimal initial planning at the beginning of the project
1c	*Absence of cascading phases*	No cascading phases which limit agility
1d	*Continuous process improvement*	Presence of continuous process improvement mechanisms
2	***Risk Reduction***	***Techniques to reduce risk without reducing agility***
2a	*Emergent plans, architecture, and design*	Methodology creates and refines plans, architectures, and designs as work evolves and requirements are understood
2b	*Engineering practices*	Integrated engineering practices can reduce risk by continually validating code through pair programming, continuous testing, and daily builds
2c	*Inspections*	Frequent inspections of product and code
2d	*Development methods*	Process is provided to reduce risk and increase quality
2e	*Specification analyses*	Provide mechanisms to analyze requirements specifications
3	***Focus on working software***	***Primary focus on building software with tools only used to support thinking and collaboration***
3a	*Artifacts minimized*	Methodology minimizes the creation of artifacts
3b	*Working software is the primary measure of progress*	Progress is judged primarily through evaluating the working product increment
4	***Adaptive, emergent, and self-organizing work and teams***	***Teams are presented with work and they self-organize to complete the work as the team determines***
4a	*Team self-organizes*	The methodology allows the team to self-organize and determine the best way to build the product
4b	*Emergent requirements*	Initial requirements initiate the project, and the rest of the requirements emerge as the project evolves
4c	*Team decides roles & responsibilities*	Process definitions and roles do not constrain the team
4d	*Adapts to changes in requirements*	No onerous "change management" mechanism to control emerging requirements
4e	*Skills of team members*	Skill level and versatility of team members
4f	*Adapts to changes in technologies*	Provide mechanisms to learn new technologies
5	***Customer Interaction***	***Customer and team collaborate to adaptively craft the product given time, costs, quality, and functionality***
5a	Customer involvement in project	Degree that the customer needs to be involved in project
5b	Adaptive empirical relationship with the customer	Customer and team collaborate to adjust time, quality, cost, and functionality as project evolves
6	***Iterative and Incremental***	***Working code is developed in an incremental fashion***
6a	Delivers working software products in a frequent manner	Methodology fosters the development of frequent and regular product increments
6b	Early delivery of working software	Working software must be generated from the 1st increment
7	***Management Practices***	***Empirical management of the work, based on team capabilities, cost, time, quality, and functionality***
7a	Methodology provides adaptive and empirical management practices	Use of adaptive, empirical mechanisms to manage projects. Pert charts and time reporting are not used tools
7b	Capability to measure progress by using empirical techniques	Techniques for measuring project progress. Artifacts and time worked are not used to measure progress
7c	Assess team productivity	Techniques provided to monitor changes in productivity
7d	Team practices	Techniques to improve productivity and empower teams
7e	People practices	Individual capabilities taken into account

4. Evolutionary Model with Agile Practices

ADEPT is a modified evolutionary lifecycle model that incorporates several concepts and practices from previously developed agile lifecycle models. The main motivation behind the development of ADEPT was to provide software developers at ABB with a "lighter" and more "agile" lifecycle model to develop software during the technology development phase that typically delivers "alpha" systems that will become new products for the Business Units. Even though several agile lifecycle models have already been developed, ADEPT was developed to accommodate the ABB's development culture and to incorporate proven agile practices in a modular fashion.

As explained above, ADEPT incorporates several proven agile practices from XP, Scrum, and DSDM that the author deems to have had success in the past. Several objectives behind the development of ADEPT are explained as follows:

(a) Provide developers with an iterative and incremental approach to deliver initial "alpha" systems during the technology development phase.
(b) Provide developers with a mechanism to have an adaptive response to changing and emerging requirements during the technology development phase.
(c) Reduce the number of artifacts and documents that developers require and place emphasis on the working piece of software during the technology development phase.
(d) Use frequent inspections for managing and controlling work, in an adaptive fashion during the technology development phase.
(e) Increase speed in the software development lifecycle.

Traditional evolutionary models provide an environment conducive to incremental software development. However, evolutionary models are not agile by definition. Some issues that need to be addressed to make them more agile are explained below:

- Evolutionary models tend to involve significant planning and documentation with updates to the documentation made throughout the lifecycle.
- Requirements are gathered and documented for the entire project at the early stage in the lifecycle.
- There is a significant effort addressing changing requirements in traditional evolutionary methodologies. A relatively large requirements document has to be continuously updated throughout the development lifecycle.
- User and customer involvement needs to be increased.
- The delivery speed of software in evolutionary models can be improved.
- Testing is typically performed at the end of the lifecycle and the time and effort devoted to testing the software becomes very limited.

4.1. Agile Practices in ADEPT

This section describes the areas in the traditional evolutionary model that were modified to develop ADEPT. A mapping to Table 1 is made.

Process

- *Scalability*. ADEPT has been primarily used in small development teams (4-8) but could use the Scrum approach to scale up a development team, by having "Scrum of Scrums" meeting.
- *Planning to define project*. Iteration planning, as in the Scrum approach, is used to develop a plan at the beginning of a cycle in ADEPT.
- *Absence of cascading phases*. ADEPT follows an evolutionary approach for software development and no cascading phases are followed.
- *Continuous process improvement*. Daily and weekly team meetings contribute to establish continuous process improvement culture.

Risk Reduction

- *Emergent Plans, architecture and design*. In ADEPT, design is not completed for the entire project at one time, but a design discussion is held one-iteration at a time.
- *Inspections.* ADEPT incorporates frequent testing at the end of each cycle. Three types of testing are performed. Acceptance tests are written with the customer prior development of any code. The developers

test the scripts implemented performing unit tests. Functional testing is performed independently to test final functionality of the system. Weekly reviews are held to examine risk and also to discuss the tasks completed during the prior week, the problems faced during the prior week and the tasks to be accomplished the following week. If pair programming is used, this serves as a peer review process.

- *Specification Analysis*. A small set of selected and prioritized requirements is implemented in each cycle and documented in a spreadsheet document.

Focus on working software

- *Working software is the primary measure of progress*. A working system at the end of each cycle is the primary measure of progress in the project.

Adaptive, emergent, and self-organizing work and teams

- *Emergent requirements*. New requirements are handled and negotiated between developers and users and/or customers at risk mitigation meetings.

Customer interaction

- *Customer involvement*. Customers and/or users work very close to development team participating in several meetings and providing feedback.

Iterative and incremental

- *Working software products are developed in a frequent manner*. A working prototype system is delivered at the end of each cycle.

Management practices

- *Assess team productivity*. Assessment of team productivity in ADEPT is performed by the delivery of working software at the end of the development cycle.

4.2. ADEPT Process Model

ADEPT adds agility by calling for only the most essential documentation. Two main artifacts are maintained throughout the project lifecycle: a Project Plan document and a Requirements Worksheet. Most of the project management data is maintained in a single spreadsheet to minimize documentation effort. Information pertaining to each feature is maintained in individual feature-sheets within the worksheet. Risk mitigation meetings are held weekly throughout the project to evaluate progress, manage risks, and discuss system design; status, risks, and design information from these meetings are all captured in the worksheet. ADEPT is comprised of three main phases: (1) Project Evaluation Phase, (2) Feature Development Phase and (3) Project Completion Phase. Figure 2 provides a graphical representation of ADEPT. The ovals portray process activities, curved rectangles represent the artifacts, thicker arrows indicate control flows throughout the process, and the narrower arrows indicate data flowing as a result of the activities.

During the Project Evaluation Phase, desired system functionality is captured as *features*, project scope is defined and negotiated, *features* are assigned to groups for implementation, and system requirements are negotiated with the customer.

The project team is comprised of all members working collectively on system development. The project manager oversees the project development. The project team meets with the customer to negotiate project scope; the negotiation process enables the project team to evaluate potential project completion in consideration of the resources (time, finances and manpower) available to them. The project manager divides the team into several groups, each comprised of two or more individuals. The project manager assigns the gathered features to these different groups for development. Two artifacts are produced during this phase -- a *Project Plan,* and *a Requirements Worksheet*. An overview in the R*equirements Worksheet* documents the gathered features, feature assignments to groups, scheduled project completion date, membership of the groups, and the team members' assigned responsibilities. The project risks, addressed during weekly risk mitigation meetings, are also documented in the worksheet, or in a stand-alone Risk Sheet. Several activities are performed in the Project Evaluation Phase. These are explained below.

- An initial meeting between the customer and the developers is held to gather desired system characteristics, which are collected as *features*.

- The collected features are analyzed to determine project scope, project risks, and required resources for project completion.

- The project manager divides the team equally into groups of two or more individuals.

- Before starting Feature Development, the project team negotiates the scope of the system requirements with the customer at a very high level.

During the Feature Development phase, features are planned, designed, implemented, tested, and integrated with the overall system. At the beginning of this phase, the groups elicit requirements for the assigned features and analyze feature-scope; additionally each group, with project manager's lead, meets with the customer to negotiate and clarify requirements for the assigned features. The groups also write acceptance tests for each feature-requirement before writing any code. At this point the customer prioritizes each elicited feature-requirement and reviews/approves the acceptance tests; the acceptance tests and the feature priorities are documented in the Requirements Worksheet. After the negotiation activity, each group concurrently employs ADEPT to develop the assigned features; the actual completion date of each requirement is also documented in the Requirements Worksheet. The initial prototype is ready for customer evaluation at the end of first Feature Development phase. The customer evaluates the prototype system via the acceptance tests, and suggests modifications, if any. The customer evaluation activity allows the project team to receive customer feedback/suggestions at the end of every Feature Development phase. To ensure customer satisfaction and software quality, the Feature Development phase is iterated to incorporate customer-requested modifications or additions, until the customer finally approves the evolutionary prototype system. Several activities are performed in the Feature Development Phase. These are explained below.

- Each development group elicits requirements for the assigned features from the customer and group members further evaluate these features and their requirements, to determine if the features can be implemented with the available resources (manpower and time). After performing resource evaluation, each group, with the project manager's lead, individually meets with the customer to negotiate the features' requirements.

- After the feature requirements are negotiated, feature planning and design are performed during weekly risk mitigation meetings.

- ADEPT recommends pair programming for implementation, as a technique to improve productivity and quality.

- In ADEPT, quality is addressed in several ways; one way is via testing techniques including feature testing, functional testing and acceptance testing. It is recommended to write acceptance and unit tests before coding.

- The customer evaluates each version of the prototype system via the acceptance tests to ensure the satisfaction of all system requirements.

Customer satisfaction and software quality is achieved by performing iterations on the Feature Development phase to incorporate customer-requested modifications, until the customer approves the evolutionary prototype system. The third phase of the model is called Project Completion Phase and it has two activities.

- The customer validates the completed prototype system. After final customer validation, the prototype system is packaged and delivered.

- After the system delivery, the development team evaluates the positive and negative experiences during the project. The goal is for the team to incorporate the positive experiences and conduct planning to avoid the negative experiences on future projects. The experiences and resulting plans are documented in a Final Project Report.

Table 2 presents a summary of the ratings presented by Schwaber [11] and includes ratings for the new agility characteristics added and the estimated agility values for ADEPT.

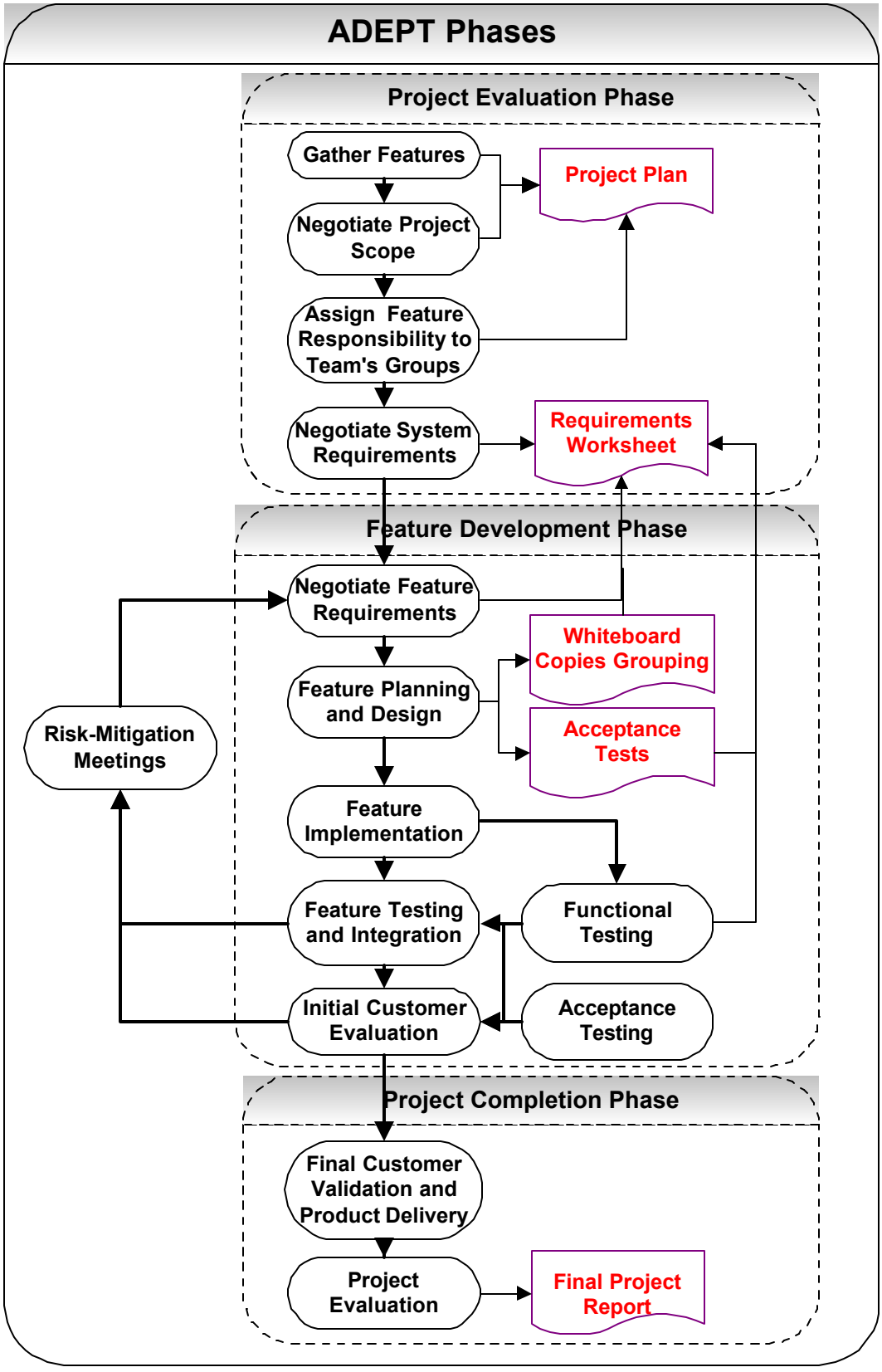

Figure 2. ADEPT Process Model

Table 2. Comparison of Agility Characteristics of ADEPT and Other Agile Models (adapted from [11])

No	Characteristic	XP	Scrum	DSDM	AGEM
1	***Process***	**2.5**	**3.0**	**1.7**	**3.0**
1a	*Scalability*	Not scalable	Scalable – Scrum of Scrums	Scalable	Scalable - Scrum of Scrums
1b	*Minimal planning to define projects*	Card-box plan	Product backlogs - sprints planning	Feasibility Study	Product backlogs iteration planning
1c	*Absence of cascading phases*	Yes	Yes	No	Yes
1d	*Continuous process improvement*	Self-mentoring team	Daily Scrum meetings	No	Daily team meetings
2	***Risk Reduction***	**3.0**	**3.0**	**2.6**	**2.6**
2a	*Emergent plans, architecture, and design*	Yes	Yes	OK	OK
2b	*Engineering practices*	Yes	OK	OK	OK
2c	*Inspections*	Test cases and functional tests	Review at the end of each sprint	Testing in iterations	Testing at every cycle
2d	*Development methods*	Paired progr. - Open spaces	Open spaces	Nothing special	Nothing special
2e	*Specification analyses*	Prioritized user stories, re-factoring	Small set of requirements every iteration	Feasibility study	Small set of requirements every iteration
3	***Focus on working software***	**3.0**	**3.0**	**2.0**	**2.0**
3a	*Artifacts minimized*	Yes	Yes	Partially	Partially
3b	*Working software is measure of success - incremental*	Working Sw at each increment	Working Sw at each sprint	Functional design/build	Working Sw at each iteration
4	***Adaptive, emergent, self-organizing work and teams***	**3.0**	**3.0**	**1.6**	**1.6**
4a	*Team self-organizes*	Team responsible	Product backlog tasks	Project leader	Project leader
4b	*Emergent requirements*	Re-factoring	Daily Scrums, backlogs - sprints	Handled at iterations	Handled at risk meetings
4c	*Team decides roles & responsibilities*	Yes	Yes	Project leader	Project leader
4d	*Adapts to changes in requirements*	Yes	Yes	OK	OK
4e	*Skills of team members*	Very high	High	Medium	Medium
4f	*Adapts to changes in technologies*	Open spaces and pair programming	Open spaces	OK	OK
5	***Customer Interaction***	**3.0**	**3.0**	**2.5**	**2.5**
5a	Customer involvement in project	At site with developers	At site with developers	Highly involved	Highly involved
5b	Adaptive customer relationship	Yes	Yes	OK	OK
6	***Iterative and Incremental***	**3.0**	**3.0**	**2.5**	**2.5**
6a	Working products frequently	Each iteration	Each sprint	Each sprint	Each iteration
6b	Early delivery	Yes	Yes	OK	OK
7	***Management Practices***	**3.0**	**3.0**	**2.6**	**2.2**
7a	Adaptive and empirical	Yes	Yes	Yes	OK
7b	Measure progress empirically	Yes	Yes	Yes	OK
7c	Assess team productivity	Yes	Yes	Yes	Yes
7d	Team practices	Yes	Yes	OK	OK
7e	People practices	Yes	Yes	OK	OK
	Agility Average	**3.0**	**3.0**	**2.21**	**2.3**

5. Conclusions

Constantine [5] provides an analysis of the pros of agile development lifecycle models. Perhaps the biggest selling point of the agile models is their lightness. Agile lifecycle models seem to be easier to learn and master. Agile processes consolidate procedures and decrease potentially de-motivating work and overhead in projects. Agile processes concentrate in delivering working and high-quality code. Design is on the fly and as required. Index cards and whiteboard sketches often take place of design documents. Early results and noticeable progress are other benefits of agile processes. Agile processes focus on short release cycles that produce a fully functional code. The simple philosophies of agile processes seem to translate well into practice.

Similarly, Constantine [5] identifies some risks and shortcomings of agile methods. These are summarized as well. All of the agile models put a premium on having top-quality people and they work best with first-rate, versatile and disciplined developers who are highly skilled and highly motivated. Agile models do not readily scale up beyond a certain point. The general consensus is that a single, collocated team with size of 12 to 15 developers is the workable upper limit for most agile processes. This is particularly true for XP. All agile models are based on incremental refinement or iterative development in one form or another. Within the prescribed short release operations of agile processes, a small team can do only so much. Some projects cannot be broken down into 30-90 day increments for practical purposes. Agile models seem to be at their best in applications that are not GUI intensive, because they rely very heavily on testing and this activity is labor-intensive and time-consuming.

From the perspective of ABB, agile development lifecycles have been studied and the current approach is to embed agile practices into the more traditional software development lifecycle models. This is the case of ADEPT.

ABB software projects are often developed as smaller projects that require fast execution and cannot afford the execution, planning, and administration costs associated with heavier weight lifecycle models. Agility can be implemented at the project or product level, as it can focus on what is necessary for the success of the project. There are many projects with emergent business requirements or unstable technology that are good candidates to include agile practices as in the case of ADEPT.

Acknowledgements

The author is grateful to the contributions to this work of Hema Srikanth.

6. References

1. K. Beck, “Extreme Programming Explained: Embrace Change”, Addison-Wesley, Don Mills, Ontario, 2000.

2. B. Boehm, “Get Ready for Agile Methods, with Care”, *IEEE Computer*, January, pp. 64-69, 2002.

3. A. Cockburn, “Balancing Lightness with Efficiency”, Cutter Consortium, September 2000.

4. A. Cockburn, A., J. Highsmith, “Agile Software Development: The People Factor”, *Computer*, November, pp. 131-133, 2001.

5. L. Constantine, “Methodological Agility”, *Software Development*, Vol., 9, No. 6, pp. 67-69, 2001.

6. L. Constantine, L. Lockwood, “Software for Use”, Addison-Wesley editors, 1999.

7. M. Fowler, “Put Your Process on a Diet”, *Software Development*, Vol. 8, No. 12, December, pp. 32-36, 2000.

8. M. Fowler, and J. Highsmith, “The Agile Manifesto”, *Software Development*, August, vol. 9, no. 8, pp. 28-32, 2001.

9. L. A. Griffin, “A Customer Experience: Implementing XP”, XP Universe, July 2001.

10. J. Highsmith, and A. Cockburn, “Agile Software Development: The Business of Innovation”, *Computer*, September, pp. 120-122, 2001.

11. K. Schwaber, “Will the Real Agile Process Please Stand up?” *E-Project Management Advisory Service, Cutter Consortium*, Vol. 2, No. 8, 2001.

12. A. Weber, “Going to Extremes”, Software Development, January, pp. 39-44, 2002.