Agile Process:An Enhancement to The Process Of Software Development

Ms Ramandeep Kaur[1], Mrs Manmohan Choudhary[2], Mr Rahul Mehta[3]

[1] Institute of Information Technology& Management ,Janakpuri, New Delhi, India

[2] Institute of Information Technology& Management ,Janakpuri, New Delhi, India

[3] M/s Billionix, New Delhi, India

ABSTRACT
In last decade, various agile methods have been introduced and used by software industry. It has been observed that many practitioners are using hybrid of agile methods and traditional methods. Agile was created in large part in reaction to the predominant waterfall model, and to a lesser extent to all "traditional" methodologies. Thus, there is strong need of agile software development life cycle that clearly defines the phases included in any agile method and also describes the artifacts of each phase. The generalization of agile software development life cycle provides the guideline for average developers about usability, suitability, applicability of agile methods. This paper presents a coherent strategy for continuous integration of some good features of classical / traditional methods of software development with the agile methodology viz., Extreme Programming, Scrum etc. to enhance the current software development techniques. This inturn can be used in a highly adaptive software environment.

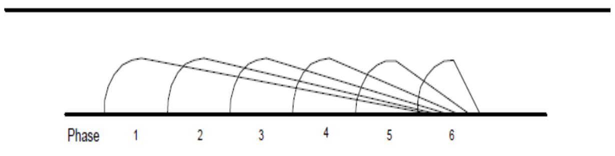
Keywords
Agile software development, extreme Programming , Adaptive software environment, Scrum, Continuous Integration.

1. Introduction

Agile methods generally promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership
philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices intended to allow for
rapid delivery of high-quality software, and a

business approach that aligns development with customer needs and company goals. Agile Methods (AMs) have been adopted by many IT organizations and have generated many quality
products of software industry. These methods have gained higher edge on traditional software development by accommodating frequently changing requirements in high tight schedules.
Agile software development is a conceptual framework for undertaking software engineering projects that embraces and promotes evolutionary change throughout the entire life-cycle of the project. There are a number of agile software development methods, such as those espoused by The Agile Alliance. Most agile methods attempt to minimize risk by developing software in short time boxes, called iterations, which typically last one to four weeks. Each iteration is like a miniature software project of its own, and includes all of the tasks necessary to release the mini-increment of new functionality: planning, requirements analysis, design, coding, testing, and documentation. While an iteration may not add enough functionality to warrant releasing the product, an agile software project intends to be capable of releasing new software at the end of every iteration. At the end of each iteration, the team reevaluates project priorities.

Iterative approach: Overlapping phases of development



Source: Adapted from H. Takeuchi and I. Nonaka, "The New New Product Development Game", Harvard Business Rev., Jan. 1986, pp. 137-146.

Agile methods promote an iterative mechanism for producing software, and they further increase the iterative nature of the software lifecycle by tightening design-code-test loop to at least once a day as opposed to once per iteration. Agile visionary Kent Beck challenged the traditional cost of change curve evidenced by Barry Boehm over twenty years ago. Beck's model espouses that the cost of change can be inexpensive even late in the project lifecycle while maintaining or increasing system quality. Beck's idealistic "flat" cost of change curve has since been revised and softened by Alister Cockburn and Scott Ambler to reflect modern corporate realities. Nevertheless, Agile ideals can be applied to reduce the cost of change throughout the software lifecycle even if the cost of change is not perfectly flat .

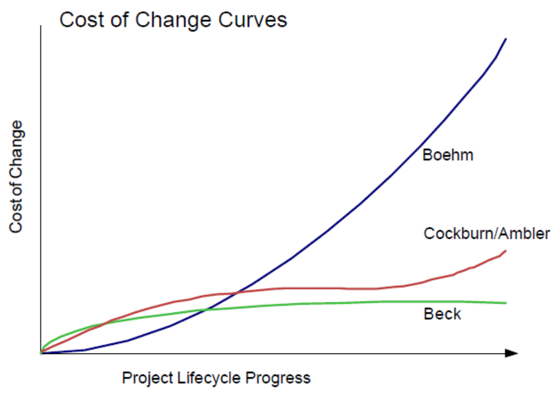
Manuscript received July 5, 2012
Manuscript revised July 20, 2012

2. Agile Software Development Life Cyclle(ASDLC)

Agile Software Development Life Cycle (ASDLC) has been designed on the basis of common practices and principles used in all existing AMs. It has various phases in it and activities performed in each phase along with artifacts required in each phase. Complete ASDLC is shown in Fig. 1 and discussed as follows:

A. Vision and Project Approval

ASDLC starts with the vision or inception phase that deals with the need of new system by analyzing problems in existing system. Management, product manager, users and team
members establish the scope and boundary conditions of proposed system. At this level, objective is apparent but the features fulfilling the objectives may be uncertain. Main objective of this phase is to identify critical uses of the system, level of uncertainty of the system, overall estimation of size and duration of the system using algorithmic or non algorithmic approach. Further, systematic analysis is performed to identify the feasibility of the system at operational and economical level with clear specified requirements. It is concerned with technical possibility of the system with incurring risk associated with it. At same level, feasibility of particular AM is assessed. This assessment is based on project type, and personnel and organizational issues etc. Business study of the system is required to analyze the essential characteristics of the business and technology. For example, a website for income tax submission must require its technicalities involved in it. Major objective of business study is identification of class of affected users. This affected class of users is useful source of information in software development cycle. It has been noticed that early estimation is useful in project approval. It is a non iterative phase and generally completed in two three weeks time. High level description of the system, early estimates are mandatory documents produced in this phase.



B. Exploration Phase

Exploration phase is an iterative and incremental phase to reduce the uncertainty and ambiguities in requirements by continuous meeting of stakeholders in the form of workshops and brainstorming. Some of the AMs have preferred customer as team member but proposed ASDLC recommends the maximum communication between team and customer to resolve the requirement related issues by using any preferred mode of communication between customer and team. Requirements may be captured in form of stories and documented in story cards that can be referred for future references. Typical format of story cards contains information about author, story id, story description, further changes in story and details of related stories etc. Artifacts produced are informal requirements description in the form of stories. Team starts with selected experienced team members on agile software development. Selected team members start communicating with the customers to understand the problems and requirements of the proposed system. Generally, while experienced team members are working on requirements, process and technology used for training and enhance the ways to improve quality of product being developed.
Further,feedback of the last release is also accommodated in this phase and major changes in the last releases are defined as new requirements.

C. Iteration Planning

Iteration planning is most important phase
of ASDLC and possesses many activities of software development required to schedule the project. First activity in this respect is review of the working software released in last iteration. Participants assess the progress and increment of the work product and discuss the future plan of the project. At the same time, requirement prioritization is performed to get maximum ROI from working software. In iteration planning, list of requirements in stack is updated depending on the feedback and requirements received from customers. This list is reviewed for prioritization of requirements. Prioritization is based on various factors mainly; value, knowledge, financial returns etc. For example, a feature that requires team to improve their technical skills has been developed in later stage but a feature that has higher financial returns must be kept at higher priority. This prioritization stack is useful in increasing ROI and producing working software in shorter time period. Prioritization has been done for only those features that are clear and unambiguous. Project manager, customer representative and team members sit together to decide the priority of requirements. Moreover, iteration plan phase possesses iterative estimation activity to estimate size, cost

and duration of the project. It also re-estimates efforts depending on team velocity. This phase also ensures the resource requirements of the system. Artifacts produced in this phase are prioritized stack requirements and set of requirements from the stack is selected for current iteration.

D. ADCT Phase

This phase is an iterative phase that deals with Analysis, Design, Coding, and Testing (ADCT). In this phase, functionality of the system is produced and enhanced in new increments. It requires several iterations before releasing the product. Decided schedule in iteration planning is decomposed in several small iterations of one to four weeks. First iteration develops the architecture of whole system by enforcing the selection of stories that form the system. In successive iterations, designing and coding along with testing is performed. In last iteration, product is ready to deploy at customer site. It incorporates designing and coding with unit testing using the concept of pair programming. ASDP always possesses simple design to incorporate changes in the requirements. Design guidelines include metaphors, CRC cards, Spike solution and re-factoring. CRC card is an index card that is used to represent responsibilities, relations of classes used in designing a particular story. Spike solution is small focused effort used to explore solution to the problem. It has been observed that adding more functionality in early stages of the software leads to a poor design document. For example, system can work for any database. This type of independency of code and design provide lesser burden when changes are triggered. Thus, ASDP use design patterns to maintain low coupling and high cohesion among modules. Functionality testing and rigorous integration testing is performed by team of customer and developers before release the product. Main activities of this phase are simple designing, maintaining coding standards and rigors testing by Test Driven Development (TDD) and functional testing. Extra care is taken to design a code simply by code and data re-factoring. Major artifacts in this phase are design documents and codes of system.

E. Release Phase

This phase can be decomposed in two sub-phases
namely; pre-release and production as shown in Fig 1. Pre-release phase recommends extra testing (i.e. integration and acceptance testing) and checking of functional and nonfunctional requirements of the system to be released. It has
been advised to include some minor changes expected by the user in the release and major changes are expected to accommodate in next iteration. On the other hand, production phase deals with releasing the product for customer use. At this time, training for users of the system is provided for operation ease. It has been observed that team handles two responsibilities after first release of the system. Firstly, team is involved in enhancing the functionalities of product. Secondly, team has to take responsibility of system in running state thereby providing customer helpdesk. We have attempted to define the ASDLC after reviewing the all phases of software development of existing AMs. We have also included the phases introduced by other researchers thereby increasing the trust and faith on agile software development.

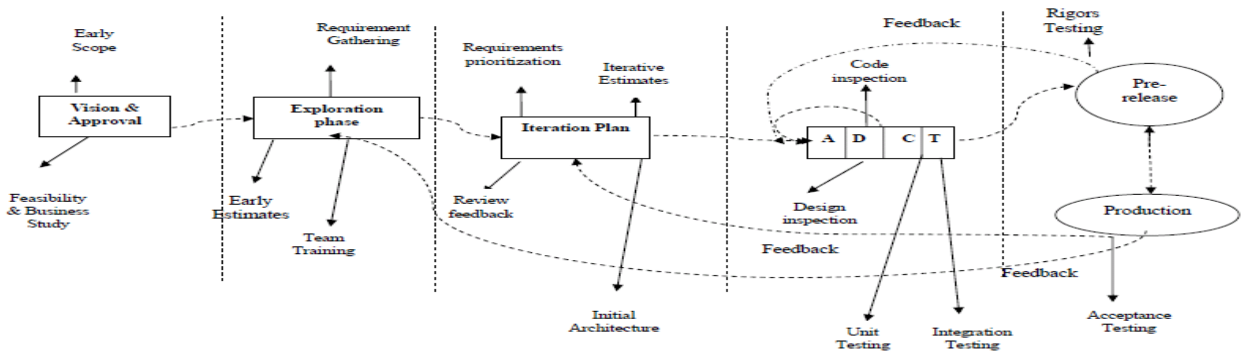


Fig. 1 Agile Software Development Life Cycle

3. The Agile Methods

Adaptive Software Development (ASD) is based on Complex Adaptive Systems theory and treats software development as a collaborative learning exercise. ASD is based on the “Adaptive Life Cycle” (which continually cycles through three phases named “Speculate,” “Collaborate,” and “Learn) and the “Adaptive Management Model” (also called “Leadership-Collaboration” management).

 Dynamic System Development Method (DSDM) is not properly a “method” because it does not provide guidance about how development projects should be run. Rather, it is mainly a philosophy about system development that consists of nine principles. DSDM focuses on system development and does not get into the details of writing software, so it can be used in conjunction with any of the more software-intensive Agile methods, like XP.

 Extreme Programming (XP) is a collection of 12 practices that focus specifically on the mechanics of developing software. These practices include such topics as The Planning Game, Pair Programming, Refactoring, and Testing.

 Feature-Driven Development (FDD) treats software development as a collection of features that are implemented one at a time.

 Lean Software Development (LD) is not really a software development method. Based on the principles of lean manufacturing, LD provides a set of seven principles for making software development more efficient, and it amplifies those principles with 22 tools.

 Scrum is primarily a product development method. Its seven practices focus on planning and managing a development project but do not address any specifics about software. Therefore, it can be used in conjunction with any software development method.

4. Agile Development and Teamwork

A Clear, Elevating Goal

Iterative and incremental development along with user collaboration plays a central role in keeping the goal visible and clear. The usage of incremental development allows the developers and the customer to define and work on smaller but clear goals. An important part of the increment is the definition of—often automated — acceptance tests. Their definition is what really makes the goal clear to the team. The usage of, preferably short, iterations allow the team to have the feedback necessary to understand if what they have done is what the customer expected. In fact, acceptance tests are very helpful, but the experience of using the software gives the customer a better understanding of her needs. This, often, leads to a refinement of the goal without a loss in clarity.

A Results-Driven Structure

Agile teams are structured in order to deliver valuable software on time and on budget in a context of frequent changes in requirements. An effective team structure has four necessary features

1. Clarity of roles and accountabilities: For agile development some of them are defined by means of the rights that the customer and the development team have.

2. Effective communication system: Agile development puts an emphasis on face to- face communication .The team members tend to be located close to each other, possibly in the same room, so the speed of communication is optimized; the customer is encouraged to interact closely with the developers, so the feedback loop is shortened and the goal remains visible and clear.

3. Monitoring individual performance and provide feedback: In agile development this is a consequence of the high level of interaction between the parties involved. If someone is not doing his best, this becomes very clear very early to everybody.

4. Fact-based judgments: Agile development methods submit all activities and products—including software—to a usefulness test: they must contribute in some way to the achievement of the goal, otherwise they are dropped. The process is streamlined by executing only the activities that simplify the work of the team. Documentation is written only if there are people willing to read it. Software is kept as simple as possible, so it is easier to change. Future extensions will be examined when the need will arise. Gold plating is loathed and avoided. Code quality is kept as high as possible. Finally, technology is used only when necessary—very often using a whiteboard or CRC cards during a design session is more effective than using the latest CASE tool.

All these techniques allow the development team to travel light and focus only on what matters for the achievement of the goal.

Unified Commitment

Agile methods tend to involve the entire team in all phases of development. The customer is in close contact with the team, so everybody can better understand the requirements, and have the goal clear. Design sessions make extensive usage of techniques such as CRC cards and whiteboards that bring the whole team together to discuss design and programming issues, in which everybody can give a contribution. All these things help greatly in fostering unified commitment.

External Support and Recognition

Agile methods recognize explicitly the importance of external support. In fact one of the principles of the Agile manifesto states “build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.” They recognize that,without appropriate resources, software development is simply not possible.

4. ADVANTAGES OF AGILE

- Agile methodology has an adaptive team which is able to respond to the changing requirements.
- The team does not have to invest time and effort and finally find that by the time they delivered the product, the requirement of the customer has changed.

- Face to face communication and continuous inputs from customer representative leaves no space for guesswork.
- The documentation is crisp and to the point to save time.
- The end result is the high quality software in least possible time duration and satisfied customer.

In a nutshell this means that you can get development started fast, but with the caveat that the project scope statement is "flexible" and not fully defined. Hence this can be one of the major causes of scope creep if not managed properly.

5. Challenges with Agile Methodology

- In case of some software deliverables,
- especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
- There is lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is not clear what final outcome that they want.
- Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.

6. Ever Changing Scope

1. Traditional Software Development

Processes are based on scenarios that are envisioned in the planning phase. New scenarios, or changes to existing ones are often considered "bad" for the project plan. Change in scope lead to slippage of schedule, increase in cost and lot of frequent changes to the project plan baseline.This happens because scope, cost and schedule baselines are predicted early on in the planning phase. Project is “expected” to stick on this baseline there after. Changing the baseline plan is an option but it is not very desirable neither is it always possible (especially late in the project).

2. Agile Approach

Agile model welcome change. Change is looked at as the driving force towards delivering a better product/ service. Change and new requirement help to define the deliverable for the next iteration. Changes fuel incremental growth of the application in a direction desired by the customer. It drive the design and development. Customer is always happy with the end product because he has had active participation in the development process. Cost and time baselines are fixed in the beginning of the project. Scope is never base lined and is added incrementally. Agile process encourages the business to learn about their need as the software is build.

References

[1] Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile Software Development Methods: Review and Analysis. VTT Publications 478

[2] Abrahamsson, P., Salo, O., Ronkainen, J., and Warsta, J., “New Directions on Agile Methods : A Comparative Analysis”, In Proceeding of 25th International conference on Software engineering 2003.

[3] Aoyamma, M., “Agile Software Process and its Experiences”, In IEEE Transaction 1999.

[4] Bhalerao, S., and Ingle, M., “Formalizing Communication Channel in Agile Methods”, In Proceedings of International Conference on Trends in Information Science and Computing (TISC07), Dec. 2007.

[5] Cockburn, A., Agile Software Development, Reading, MA: Addison-Wesley, 2002, p. 215.

[6] Craig Larman, “Agile & Iterative Development: A Manager’s Guide”, Addison-Wesley, 2003.

[7] Highsmith, J., Agile Software Development Ecosystems, Addison Wesley, 2002

[8] Pressman, R., Software Engineering A Practitioner Guide, McGraw- Hill 6th Edition.

[9] Sommerville, Ian (2007) [1982]. "4.1.1. The waterfall model". Software engineering (8th ed.). Harlow: Addison Wesley. pp. 66f..

Ramandeep Kaur, currently working as an Assistant Professor received Bsc(Hons), MCA from Delhi University &Guru Gobind Singh Indraprastha University in 2006 & 2009 respectively. She has worked for 6 months as a Software Programmer at National Informatics Center, Yojna Bhavan, New Delhi. She has also worked as a Lecturer(IT) in IITM,Delhi for 1 year. Her area of interests are Software Engineering, Computer Networks, Network Security, Java, Operating System, Asp.Net,Front End Design Tools, etc.

Manmohan Chaudhary ,currently working as an Assistant Professor , received B.A(Economics) & MBA(Marketing) from Guru Nanak Dev University, Amritsar. She has also worked as a Lecturer in IITM,Delhi for 1 year.

Rahul Mehta ,currently working as a Senior Trainer in M/s Billionix received BCA , MCA from Guru Gobind Singh Indraprastha University in 2006 & 2009 respectively. His area of interests are Software Engineering, Computer Networks, Network Security, Java, Operating System, Asp.Net,Front End Design Tools, etc.