Agile Software Development Methodology and its Cost Estimation Technique

Dr. Utpal Roy, Susmita Das

Abstract: This study bridges the gap between the software Industry and Academia through improvisation of some technique into one methodology. The main objective of this article is three-fold: 1) to focus upon development practices used by small software firm. 2) to study the documentation practices in small software firm and suggest one type of documentation practice which will be beneficial for the current practice 3) to find the solution of problem faced by the small firm while materializing the requirement story board drawn by the client into cost estimation. Together with these a simple and easy to use empirical formula for the evaluation of the cost for the Agile development of a medium-size software has been proposed from the past resultant data of development.

Keywords: Agile, SDLC, Software Development, Extreme Programming, SCRUM

1. INTRODUCTION:

Agile software development is a group of software development methodologies based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change. It is a conceptual framework that promotes foreseen interactions throughout the development cycle.

The Agile Manifesto[1] introduced the term in 2001. Incremental software development methods have been traced back to 1957[2]. In 1974, a paper by E. A. Edmonds introduced an adaptive software development process[3]. So-called lightweight software development methods evolved in the mid-1990s as a reaction against heavyweight methods, which were characterized by their critics as a heavily regulated, regimented, micromanaged, waterfall model of development. Older practitioners Proponents of lightweight methods (and now agile methods) contend that they are a return to development practices from early in the history of software development [2].

Early implementations of lightweight methods include Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These are now typically referred to as agile methodologies, after the Agile Manifesto published in 2001[4].

The Agile Manifesto [1] stresses the importance of interactions and peoples over processes and tools, working software instead of detail documentation, active customer participation and involvement rather than time and effort expended on

negotiating contracts and willingness ability to take changes over commitment to static plan. The overall feedback from the software organizations those have adopted the Agile development methodology is positive to great extent. The major benefits gained from the Agile development are increased productivity, expanded test coverage, reduced time and costs, understandable, maintainable and extensible code, better collaboration and higher customer satisfaction[5]. The incorporation of Agile development revealed some challenges such as slow participants buy-in, opposition to pair programming, lack of detailed cost evaluation, scope creep, reduced focus on code base's technical infrastructure and maintainability, difficulty evaluating and rewarding individual performance, the need for significant on-site customer involvement, management support, competent managers and developers, and extensive training. These findings gleaned from the case studies and experimental work such as those by Drobka, Noftz and Raghu[6], Schatz and Abdelshafi[7] and Willam, Kessler, Cunningham and Jefferies[8] provide detailed insights into the application of agile software development for specific projects.

In Today's world previous software development approaches are becoming inadequate as the requirements has become more complicated, not always easy to predict before hand and there has been found a need to have it more robust and extensive in terms of usage. Now, development has become more focused to meet the changing business needs rather than emphasizing on milestone requirements. To meet those requirements many software libraries, frameworks and tools has been invented from open source community. Even for most purposes majority of open source software can be used even for commercial usage.

Focus of the industry has been changed from producing one concrete and futuristic product to faster and easily modifiable software. Changing information technology (IT) focus has also induced use of information technology for not only big corporate houses but also individuals. Days are gone when people used to dream about their web presence, now a day it is no more a luxury but a necessity. So, naturally changing IT needs also enhanced web development companies to change their customer base from big trading houses to individuals.

Software development can be seen as a process of knowledge acquisition, in which human beings progressively learn about the intended behaviour of the desired systems. Thereby, development is subject to considerable amounts of uncertainty and variability, that make it impossible to proceed in a purely incremental fashion — at some points, the need always arises to reconsider part of the accumulated knowledge. With this problem in mind, agile development

methodologies have been gaining popularity in recent years as a means to enhance productivity, and there have been attempts to supplement them with formal techniques for better reliability.

In the arena of extensive use of open source solutions many small and medium size software companies has took a major concern about materializing the changing business needs of people. Major concern about using such solution is easy to develop and extensive robustness in terms of web development projects. A key point of using such solution is much cheaper production cost and shorter turnaround time.

It has been seen that orthodox software development methodologies used to take such a long time to complete the full assignment, change in the requirement is very common problem to be faced at the time of completion. Moreover the designated person to raise such a requirement may not be working with the same organization to verify requirement fulfillment. Development team may change, new team member also take a long time to execute at the fullest strength as visualization of the requirement may not be very clear to them. To get rid of such indefinite problems, one quickest and easy-to-achieve development methodology has been come into picture which is ideal for the scenario is Agile Software development methodology. This methodology differs from all other traditional approaches of software development for its flexibility in nature and less emphasis on up-front and strict plan-based control principles.

The term coined by seventeen software engineering consultants who were planning independently to create change tolerant methodologies, retreated to Snowbird, Utah to discuss commonalities between their respective methodologies. They classified their methodologies as agile [1], a term with a decade of use in flexible manufacturing practices. The term promoted the professed ability for rapid and flexible response to change of the methodologies.

Rapidly growing and volatile internet software Industry and emerging mobile application environment has faced so many problems with rigid software development methodologies. These methodologies are sufferer of frequently requirement changes, in search of such a methodology which will be light weight, nimble and easy to modify the term agile software development has been coined. Some features of agile software development methodology are:

- Random opportunistic to make changes in the specification and easy to modify.
- Reflection of human role in team development rather than rigorous process maintenance.

- Focused on delivering business value in very short while thus minimizes the risk of non fulfillment.

This methodology can be easily adopted as it is having capability to satisfy customer at the time of taking contract. The core emphasis is to handle inevitable changes in the requirement rather than freezing the time frame when to stop taking changes in the requirement.

There are certain constrain that mattering a lot for agile development methodology:

- Involvement of team members and the customer at every stage of development.
- Construction of small size easily cooperated team.
- Emphasis on working code rather than documentation.
- Through communication between customer and development team for a fruitful project completion.

Each phase in agile software methodology takes very short span to produce a reasonable amount of result. The span is restricted between one week and a month. The process divides requirements into short milestones rather than dividing deliverables into modules. Thus each phase looks for achieving short term goal and risk minimization.

Some current state in the method has been found in form of Extreme programming, Scrum, crystal family of methodologies, Future driven development, Rational Unified development process, dynamic system development method, adoptive software development and open source development method. Here in this paper we will discuss about two most popular agile development practices.

Organization of the article is as follows: Section 2 describes two widely used methods- Extreme Programming and scrum methodology of development, Section 3 embodies the cause analysis of choosing such methods, Normal documentation practice is presented in details in section 4, Section 5 depicts the Invention of Functional Specification Document, Comparison between FRS and SRS has been placed in section 6. Benefits of giving more emphasis on designing rather than coding and the Merits and demerits of Agile Development are in the sections 7 and 8 respectively; finally the Empirical Formula for cost estimation of the Agile development (in Scrum View) is presented in section 9.

2. EXTREME PROGRAMMING AND SCRUM

XP has been the most popular agile methodology [9], and has been widely accepted as the most important one [10-13]. At the core of XP are twelve simple practices, they are now new to the industry but putting them together with come up with a wider rate of success for the success of the methodology.

These practices are:

Planning Game: here the main idea is not to conceptualize the whole planning, but to plan as much as it feasible for the next iteration to come. Here the planning phase is strongly associated with the group work for both the customer and the development team

Short releases: Here main emphasis is to shorten the iteration span from twice a year to twice or once a month to maintain the short term goal.

Metaphor: Using a common vocabulary practices so that some terms become habitual for both the client and the developing team. Some common term helps the development team and customer to understand the requirement faster.

Simple Design: Here the main emphasis is to create design as simple as possible to ensure that software must run all tests, have no duplicate logic, state every intention that is important and have the fewest possible classes and methods.

Testing: Here mainly two types of testing are taken into consideration-unit and functional testing. The software must pass unit testing except at the time of integration of some new feature. And functional testing is considered as a parameter to confirm the progress of the project.

Refactoring: It emphasizes on restructuring the code to add new functionality keeping the functionality same as before. It is used to remove duplication and add flexibility. Here extensive unit testing are needed to ensure the functionality as before.

Pair programming: In this practice, two developers will be sitting in one computer. One must be writing the code and one must be thinking about the consequences. They may change their role every now and then to encourage and appreciate both of their work. In most of the cases it has found a positive result.

Collective Ownership: every programmer may change their code every now and then to ensure the functionality and unit testing. It will be maintained to ensure that the software will run in every condition.

Continuous Integration: After each and every development day, the effort will be integrated with the whole tested software not at the end of the development phase to ensure the new functionality has been adopted well.

40-hourWeek: Not to work more than 40 hours a week idea should have been adopted as overtime will minimize the quality of the work.

On-Site Customer: A customer should be available all the time with the team especially at the time of unit testing to ensure functionality is same as expected.

Coding Standards: Since everyone is having right modifies anyone's code. So it is better to adopt common coding practices throughout the whole development phase.

Scrum: Scrum, on the other hand, concentrates on agile project management and not on the used development methods. When talking about Scrum, it is suggested that Scrum should be adopted all at once. However, it was decided that the Scrum review, meeting and the practices will be done to produce sprints. All work is done in Sprints. Each Sprint, in general, is an iteration of 30 consecutive calendar days and is initiated with a Sprint planning meeting, where the Sprint Backlog is agreed. This is a subset of Product Backlog requirements that defines functionality to be developed in the current Sprint. Every requirement is further broken into tasks that each takes roughly 4 to 16 hours to finish. Functionality is developed by the Team, i.e. a group of developers that are collectively responsible for the success of each iteration and of the project as a whole. Here are some points which will be taken into consideration they are as follows. The estimated size for the project is only a couple of sprints. The selected practices are listed below with justifications for their selection.

Product Backlog The product backlog was selected to function as the requirement repository, which contains all the currently known tasks. The backlog will be updated whenever a new requirement is received or when additional information for an existing requirement is available.

Effort Estimation The client required approximate effort estimation for the entire project before the start of the implementation phase. The initial estimates were selected as the starting point for the project effort estimates, even though this does not exactly adhere to Scrum ideology. In pure Scrum, only the requirements for the next sprint should be estimated.

Sprint To support the XP short releases practice, the Scrum sprints were selected as the short iterations.

Daily Scrum Meeting The daily Scrum meetings were selected as the main team organizational activity. Since both the project and the project team is small, this meeting was seen as an adequate way of keeping everyone up-to-date with the current project status. For this small project, this adoption of the Scrum process was seen as an adequate set of project management practices.

3. CAUSE ANALYSIS FOR CHOOSING SUCH A METHOD:

- *Vaguely stated Requirements*-at the time of awarding a project, the requirements are somehow not very clear from both the parties, the client gives a vague idea about the requirement and the contractor agrees but the problems while actualization of those ideas were not considered at the very moment.
- *Lack of definitive requirements-Requirements at the initial stage usually not defined in a descriptive manner. Flow of project usually found to be different from the initial stage as we go on.*
- *Requirement misunderstanding-Sometimes misunderstandings are also found when it comes to actualization of the exact requirement.*
- *Missing requirements-Some requirements are found hidden in nature, like to complete a requirement the contractor may find some out of scope works are necessary to be completed before the original one.*
- *Lack of customer contact-Sometimes delay in customer feedback unnecessarily cause delay in the completion of the project. As a result when it comes to deliver the whole project, lack of intervention of customer is found to be one of the main reasons for the delay.*
- *Frequently changing requirements-There are two types of project a company works for.*
- *a)Implementation of new project and b)Maintenance of an old completed project. For Implementation of a new project, changes in requirement are more frequent in nature.*
- *Complexity of requirement documents-Sometimes it has been realized that documentation at each and every requirement change is unavoidable as some modification may found harmful for the future stage of project. And to go to the last stage proper documentation is necessary.*
- *Problem more of organizational than technical-Customer organizational hierarchy as per project responsibility allocation matrix may cause problem to the project completion.*

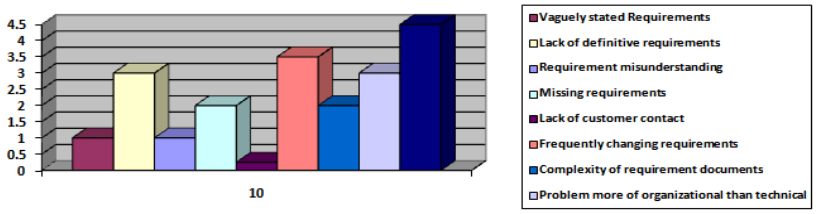
- *Document missing at user level-user level documentation missing is one of the root causes to defer the project as their expectations sometimes do not match with the actual one.*

Now if we consider the root causes and mark them as per their role in delaying the project we may represent them as follows:

[This table has been made on the basis of an interview taken with two software developers in a medium size company, around Calcutta, West Bengal, India]

Serial No.	Root Cause	Marks out of 10
1	Vaguely stated Requirements	1
2	Lack of definitive requirements	3
3	Requirement misunderstanding	1
4	Missing requirements	2
5	Lack of customer contact	0.25
6	Frequently changing requirements in	
7	Complexity of requirement documents	2
8	Problem more of organizational than technical	3
9	Document missing at user level	4.5

From the above table we can conclude that lack of proper documentation is one of the important root cause found for delay or differing a project. Our main emphasis is here to discuss and further establish the fact in more constructive manner throughout the article.



Agile software development methodology has certain features which is somehow unique from structural process bound methodologies. The methodology is more people bounded rather than process. According to Cockburn[11] there should be a minimal documentation from the point of view of agile software. Maintaining necessary documentation only up to next round of revision is necessary. Until and unless second round of revision stage is prepared, we should not move with the project-here documentation is a secondary objective. The team size used to be very small. And a tied understanding and

bonding is the most important thing for the success of a project. So human importance is felt strongly than the documentation.

4. NORMAL DOCUMENTATION PRACTICE FOR INDUSTRY-SRS

Software requirement specification is very popular term among development community. Requirement specification is the one of such important document before designing software. A good software requirement specification should identify and cover the following area:

i. Define the problem - The first step always begins with clarifying the objectives, defining the concerned issues, and limiting the problem so that it can be effectively studied.

ii. Identify feasible alternatives - All the alternatives should be considered to make sure that the best approach is chosen.

iii. Select the evaluation criteria - The criteria for the evaluation process can vary considerably, so the appropriate ones must be chosen.

iv. Applying modeling techniques - A model or series of models should be used.

v. Generate input data - The requirements for appropriate input data should be specified.

vi. Manipulate the model - After data is collected and inputted, the model may be used. Analysis after using the model will lead to recommendation for some kind of action.

Software Requirement Specification (SRS) document takes care of all the above mentioned key points. It is prior information of the software to be built. The features of the software and scope, functionality, key elements, schedule, delivery time, any revisions (if at all) are all part of the SRS document. It is also one of the important aspects to be followed before designing a test case. SRS document build upon agreed terms and specifications about the software of both the customer and the development company.

5. INVENTION OF FUNCTIONAL SPECIFICATION DOCUMENT

Advancement in the documentation with more visual appeal can be a better way to communicate between the customer and the software development firm. Using visualization methodologies people are often able to understand the information presented in a shorter span of time and to a greater depth. The term “visualization” can refer to the activity that people undertake while building an

internal picture about real world or abstract entities. Visualizing can also refer to the process of determining the mappings between abstract or real-world objects and their graphical representation. As we have seen in SRS documentation, it is more useful for understanding of the developing company than the customer. The user of the software may get a rough idea of the functionality of the coming software from SRS documents but he is completely unaware of the visuals of the same. Another reason to emphasis on the visuals is research has shown us that visually observation is easier for human brain to percept anything than going by just words.

Here we can modify Software Requirement Specification document with Functional Requirement Specification (FRS). In Functional Specification document more emphasis is given on the understanding the functionality rather than the overall software requirement. Each and every functionality can be defined here with exact visuals. Here we are assuming one project where one login page may take user to open one administrator application. Now for this particular case, the functional Specification documentation for this particular page may include one visuals of a login page along with goal and level of authentication that has to be provided by the software has to be mentioned. Here level of authentication may refer to the level of user with authority to access different data. If that links to any database then the database authentication information also needs to be incorporated with exact visuals in the documentation. Database structure can be shown in a nutshell with all the important attributes of a table which may directly link with attributes of different tables.

6. COMPARISON BETWEEN SOFTWARE REQUIREMENT SPECIFICATION (SRS) AND OF FUNCTIONAL REQUIREMENT SPECIFICATION (FRS).

Software requirement specification in short is more emphasized on general features where FRS gives more stress on technical specifications. How data should be organized is not a part of SRS like FRS. Graphical representation of date as well as every stage to be appearing during execution of process is not a part of the features of SRS where in case of FRS it is mandatory to design and declare every stage in the documentation. Flow of data is not a part of SRS, in FRS it is mandatory to define dataflow. In a nutshell we may call FRS a superset of SRS where graphical representation is highly emphasized for better and easy understanding. Moreover, declaration of data set will ultimately lead to design a test matrix beforehand in a proper way which ultimately saves time as a whole.

7. BENEFITS OF GIVING MORE EMPHASIS ON DESIGNING RATHER THAN CODING:

Graphical representation of dataflow, declaration of screens to be used will lead people to better understand the dataset to be used. Another point to be remembered here is that visualization of features and dataset is easier to go through rather than conceptualization by words. While working with agile development, keeping track of every iteration is really a tough job. It can be easier to just go with designing and specifying the dataset to be used in every iteration so ultimately it will lead to save time and will help to keep as a future record.

8. MERITS AND DEMERITS OF AGILE DEVELOPMENT:

The cases, where it is tough to estimate and setup milestones beforehand can be considered as a perfect scenario for using agile software development methodology. It is by nature is more flexible and provide enough space for developer to decide upon the improvement or enhancement of functionality and flexibility. Here rather than breaking upon the whole process into modules we break the time to produce the result. Short term goals can be set as a result continuous involvement of developing team as well as client is very much important in the game. Continuous feedback from the client is expected. The process gets delayed if that feedback is unavailable from the client. Another major issue is that development team should be limited to 6-10 people. Team spirit and mindset of team member is another important aspect here. The process will get delayed and unfruitful if the team member continuously changes because the Agile development method is more people bound than process.

9. COST ESTIMATION FOR AGILE SOFTWARE

One of the main ideas of agile development is to perform *continuous integration*, in order to detect and resolve conflicts among several modular units of a system as soon as possible. Whereas this feature is well catered for at the level of programming source code, the support available in formal specification environments is still rather unsatisfactory: it is possible to analyze the composition of several modular units automatically, but no assistance is given to help modify them in case of problems. Instead, the stakeholders who build the specifications are forced to attempt manual changes until reaching the desired functionality, in a process that is far from being intuitive.

Cost of the software estimation models those are available, mainly based on some form of regression technique. Regression models have a mathematical foundation and are constructed by collecting data on completed projects and

developing regression equations relating them. In this way, many empirical estimation models like the COst COnstructive MOdel (COCOMO) by Barry Boehm[14,15], have been developed. This model suggests three modes and three classes of software projects where empirical constants are introduced based on past data for each of the organic, semidetached and embedded classes.

In the recent past there are several sophisticated and accurate[16-20] methods have been proposed in literature for the estimation of cost, effort and development time for agile software. There is no doubt about in it that these calculations are up to the mark and well accepted.

In this section a simple empirical formula for the estimation of cost for Agile software development has been proposed. It is clear to submit that the objective of the proposed study is not to compete with those accurate and well recognized techniques. There are many small moderate and large software industries in and around Calcutta, (West Bengal, India). Considering the advantages of Agile development procedures, it is needless to say, that even small software industries now a day have been aligned towards to agile development process and trying to avoid the traditional life cycle methods for software development. However, in fact, small as well as large software industries are not always happy with the Agile development technique. From the development experience and collecting data of some completed projects from few such small and medium software industries a regression based empirical formula have been estimated, here. For the statistical and other mathematical calculation a powerful statistical software Microcal Origin 5.0 has been used. The obtained empirical formula is given below

$$y = y_0 + a_1 e^{-\frac{x}{t}} \qquad (1)$$

Where the values of y_0 and a_1 are obtained from mathematical analysis and the value of t depends upon the nature software according the Bohem classification scheme. In fact from the mathematical analysis it has been come out that the values of y_0 and $a_1$are highly dependent on the number of Agile development sprint.

From statistical analysis the values of the parameters with possible errors are

$y_0 = 21 \pm 0.0$

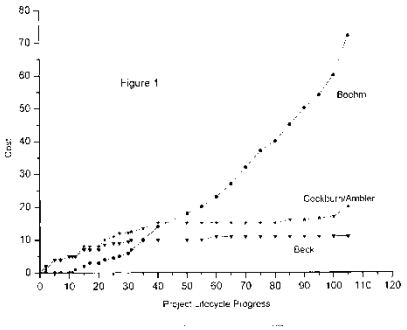
$a_1 = -21.6809 \pm 0.52538$ and

$t = 40.12025 \pm 1.82711.$

Whereas the values of χ^2=1.27196 and $R^2 = 0.9768$, having their usual meaning.

The nature of the plot is shown in Figure 2 the y axis represents the cost of development whereas x axis denoted the numbers of cycles prepared for the development of the particular software. Where as in Figure 1 nature of development curve obtained by Bohem[14, 15] with traditional method has been presented together with the results obtained by a famous agile visionary Kent Beck[16], Alister and Cockburn[17]. A close look to the Figure 1 and Figure 2 indicates that present calculation is supported as well as in gross agreement with that of others[14-17]. It should be mentioned that the points of curves in Figure 2 have been reproduced rudimentarily from the original calculation of Bohem[14,15], Kent Beck[16], Alister Cockburn[17].

The present curve does not exactly match with that of Kent Beck[16], Alister Cockburn[17], in fact, the present empirical formula was not supposed to give excellent agreement with these results as it is formulated by some mathematical approximation. But the nature of the curve produced by the empirical formula with the obtained values of the parameters tallies with that of Kent Beck[16], Alister Cockburn[17]. With increasing numbers of cycles the nature of the curve remain flat, that indicates after certain numbers of initial cycles the cost becomes linear with the numbers of cycles prepared. Though for the agile development the preparation of the cycles is most crucial part both for eXtreme progaming and Scrum view. So far our knowledge goes development cost estimation of Agile Software has not been proposed before with empirical formula.



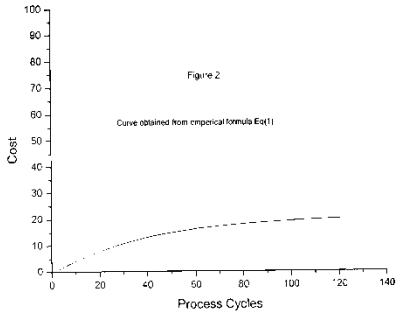


Figure 1 Figure 2

Figure 1: Nature Cost and Project Lifecycle Progress proposed by Bohem[14,15], Kent Beck[16], Alister Cockburn[17]. [*Points in the Figure have been reproduced from the respective original works.]*

Figure 2. Variation of cost with numbers of Agile Process Cycles with the help of the proposed Analytical formula in eq.(1).

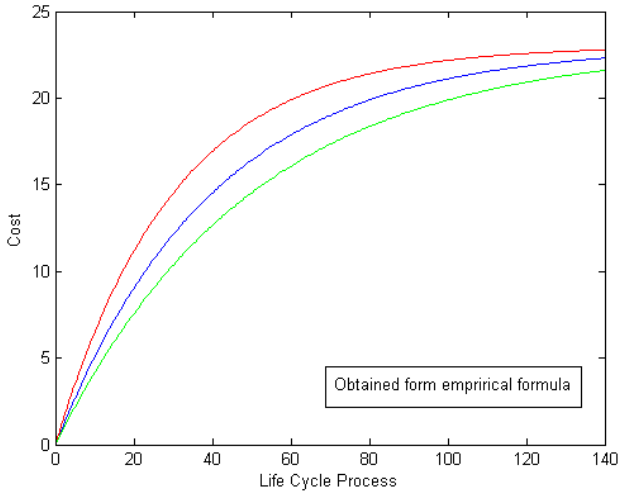


Figure 3: Nature of the curve with different values of the parameter t.

Bohem[14,15] proposed that there are three important classes of software products: ***Organic, Semidetached and embedded***. Roughly speaking, these three product classes correspond to application program, utility program and system program, respectively. In order to classify a product into any of the three proposed classes, Bohem requires to take into consideration the characteristics of the product as well as those of the development team. Data processing and scientific program are considered to be application program. Operating systems and real-time system programs, etc. are system programs. System programs interact directly with the hardware and typically involve timing constraints and concurrent processing. But according to Brooks [21] utility programs are three times as difficult to write as application programs and system programs are three times as difficult as utility programs. As stated by Brooks[21] the relative levels of product complexity for three categories of products are in ratio 1:3:9 for application utility and system programs.

Here for the Agile development process, we concentrate on Bohem[14,15] classification scheme of software. In equation (1) the value of parameter t, here, represents the class of the software according to Bohem[14,15] classification. Considering the complexity proportion of three classes of Software as proposed by Brooks[21] we have estimated values of t, for ***Organic, Semidetached and embedded*** class of software. The obtained values are

$t = 40.56,\ 50.91\ and\ 57.36$ respectively. The nature of the cost versus iteration have been shown in the Figure 3. Out of the three curves in this figure lowest one represent the Organic software, top most one represents Embedded and finally the middle one represents the Semidetached. Importantly the nature of the curves(not the exact vales) for three types of software are in agreement with that of Bohem[14,15] and Brooks[21].

10. CONCLUSION:

In this paper we have shown that betterment of normal documentation practices through using Functional Requirement Specification will be helpful while using agile methodology.

Using FRS the organization can be benefitted as it will be easier and less time consuming to document the whole process. In case the team member leaves the team this documentation methodology will be a static guideline to be used for the next user or next team member. This guideline can also be considered as a test guideline for the testing team to draw test cases also.

Last but not the least, the empirical formula can also be established through some strong case studies which in future can be a proven formula for cost estimation of agile methodology. This formula which we have found indeed will be helpful for emerging small and medium size software companies to estimate their costs. Time and effort estimation will be another area yet to be explored in this regard and is left as a next attempt for future study. We hope that this paper will be helpful for further researchers to put some business intelligence into the cost estimation so that this will be much more easy for the emerging companies to compute their cost in an easy way beforehand taking the next improvisation of the project.

REFERENCES:

1. Agile Manifesto: "*Manifesto for Agile Software Development"* http://www.agilemanifesto.org/, December 2007; Beck, Kent; et al. (2001). "Manifesto for Agile Software Development", Agile Alliance. Retrieved ,14 June 2010.
2. Gerald M. Weinberg, as quoted in Larman, Craig; Basili, Victor R. (June 2003). "Iterative and Incremental Development: A Brief History". Computer 36 (6): 47–56.
3. Edmonds, E. A. (1974). "A Process for the Development of Software for Nontechnical Users as an Adaptive System". General Systems 19: 215–18.
4. Vijayasarathy L.R. and Turk Dan, "Agile Software Development: A Survey of Early Adopters", Journal of Information Technology Management, Vol. XIX, No. 2, 2008, pp 1-8;

5. http://en.wikipedia.org/wiki/Agile_software_development
6. Larman, Craig (2004). Agile and Iterative Development: A Manager's Guide. Addison-Wesley. p. 27. ISBN 978-0-13-111155-4.
7. Drobka, J.; Noftz, D. and Raghu R. " Piloting XP on Four Mission Critical Projects.", IEEE Software, Vol 21, No. 6, 2004, pp 70-75.
8. Schatz, B. and Abdelshafi, I. " Primavera Gets Agile: A Successful Transition to Agile Development" IEEE Software, Vol 22. No. 3, 2005, pp36-42.
9. Willams L.; Kessler, R R; Cunningham, W. and Jeffries R. " Strengthening the Case for Pair Programming", IEEE Software , Vol. 17, No. 4, 2000, pp 19-25.
10. Maurer, F. and Martel, S. Extreme Programming: Rapid Development for Web-Based Applications, IEEE Internet Computing. 6(2002) p 86-90.
11. Mahnic, V. and Zabkar N. Using cobit indicators for measuring scrum-based software development, WSEAS Transactions on Computers vol. 7, 2008, 1605-1617.
12. Cockburn, A., Agile Software Development, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA ©2002; Agile Model Driven Development with UML2- By S Ambler, Camb. Univ. Press 2004(3rd Edition).
13. James E. Tomayko, An Historian's view of Software Engineering", in the proceeding of the IEEE Conference on Software Engineering Education and Training, IEEE Society Press, Los Alamitos, Ca., 2000.
14. Willams, " The XP Programmer: The Few-Minutes Programmer", IEEE Software,2003, Vol. 20, pp16-20.
15. Boehm, B.W. Software engineering economics, Prentice Hall, Englewood Cliffs, N.J. 1881.
16. Boehm, B.W. Software engineering economics, IEEE Transaction on Software Engineering Economics, IEEE Transaction on Software Engineering, 10(1):135-152, Jan. 1984.
17. Kent Beck, " Extreme Programming Explained : Embrace Change," Addison-Wesley, 2000
18. Alistair Cockburn, Re-examining the Cost of Change Curve, year 2000, , XP Magazine, September 2000.
19. Scott Ambler, Examining the Cost of Change Curve,; Agile Modeling Essays expected from the book "The Object Primer, 3rd ed.: Agile Model Driven Development with UML2", by Scott Ambler, Cambridge University Press, 2004.
20. Martin Flower, " Is Design Dead?", http://martinflower.com/article/designDead.html, 2004.

21. I. Nonaka, H. Takeuchi, " The New New Product Development Game", Harvard Business Review, January 1986, pp.137-146.
22. Brooks F., The Mythical Man-Month, Addition-Wesley, Reading, Mass., 1975.

AUTHOR'S PROFILE

Dr. Utpal Roy did his graduation and post graduation from Visva-Bharati, Siksha-Bhavana (Institute of Science). Throughout his career he secured First Class in all public Examinations. Subsequently he did his Ph.D. from the Department of Mathematics, Visva-Bharati, Santiniketan. Immediate after the completion of his Ph.D degree in 1994 he joined the Department de physique, University Laval, Quebec Canada as a Post Doctoral Fellow under Natural Science and Engineering Research Council of Canada. Later on in 1996 he joined Indian Association for the Cultivation of Science as a Senior Research Associate (CSIR). He joined Visva-Bharati as a Lecturer in Computer Science in 1997. After that he spent more than a year as a Visiting Fellow in the Academia Sinica, Taipei, Taiwan. His Field of Research is very versatile and he published almost 40 Research Papers in National and International Journals and Conferences. He became an Associated Professor in the Department of Computer & System Sciences, Visva-Bharati.