Software Development Process Models Comparison And Assessment Of Degree Of Agility Based On Agile Practices And Performance Implementation On XP And Scrum

Rishi singh, Sowmay Vijayan, V. Ilango*, A. Abdul Rasheed

Abstract: : Software Process Improvement is generally regarded a key to economic success by increasing the quality of software systems, accelerating time-to-market and decreasing development costs. The new software development paradigm is much different from the traditional approach. It is necessary to study and analyze the gap between classical and modern software development methods. The use of agile methods is becoming widespread in the system development industry. Agile methods have several benefits over traditional plan-based methods, in particular their ability to handle projects where requirements are dynamic Organization's varying needs and environments results in context specific adjustments of agile methods. In the present digital transformation world, agility processes, practice and degree of agility need to be assessed. The assessment of agile work is often based on how well the organization complies with a commercial method. In the last few years, a number of agile software development methods have been developed but a detailed evaluation (which is essential) of these methods is not available. In this paper, we survey and compare traditional software development models and agile software models, described their advantages and disadvantages, and discuss the features they inherit. The aim is to identify how gaps between practice and priority can be measured. A detailed study has been conducted to measure the perceptual performance of five Scrum practices with consideration to the prioritization of seven agile aspects. The results shows that the studied agile aspects are considered essential in the organization and that practices in some cases underperform or over perform. This paper presents a detailed comparative analysis of two well-known agile methods, XP and Scrum, based on agility characterization. However, in this study, compliance of practices to strategic priorities is considered.

Index Terms: Agile models, Degree of Agility, Extreme Programming, Process model, Scrum, Traditional software models, Software Paradigm.

1. INTRODUCTION

THE Software Evolution is recognized as an inevitable nature of a software system. The software system should be maintained to keep their functionalities along with the environment changes such as organization changes, middleware changes and so on. A new software system is typically implemented with existing software, so called reuse based development. With the history of miseries of low reliability and delayed schedule of software development, we have been studying the importance of readability, maintainability, extensibility and so on. The fact that software life-cycle is transmigratory prevents from applying a single technology to improve the overall software quality. The rapid expansion of the IT world invites new comers in software development community. Web technology has been invented as a tool for information retrieval. Recently, so many systems, including enterprise systems, whose reliability is very important, have been implemented as Web applications. Thus, a natural question arises: Are Verification and Validation techniques established in the half-century history of Software Engineering research applicable for this situation?

- *Mr. Rishi Singh and Ms. Sowmya Vijayan are currently pursuing Master degree program in Computer Applications in CMR Institute of Technology, Bengalore.*
- *Dr. V. Ilango is currently working as a Professor in the Department of Computer Applications, CMR Institute of Technology, Bangalore, India.*
  E-mail: ilango.v@cmrit.ac.in
- *Dr. A. Abdul Rasheed is currently working as Professor in the Department of Computer Applications, CMR Institute of Technology, Bangalore, India.*
  E-mail: Abdul.r@cmrit.ac.in

The variety of the target systems requires the evolution of not only the software itself but also Software Engineering. The main goal of this paper is to research and study different agile methods and find out the best agile practices through evaluation and comparison. Agile development emphasizes on creating workable software. Traditional software development methods emphasize more on documentation, but agile software development is a new way of building software where documentation is not a big issue. Agile development relies heavily on customer involvement in development process. This paper presents all key aspects of agile methods and analyses important agile practices which may be effective to select the suitable agile methods for a development team or organization. This paper is organized into four parts. Part one describes detailed study about traditional software development process models. Part two expressed about different agile development process models. Part three explore comparison between traditional and agile models. Part four experimental observation and result of agile XP and Scrum methods are discussed.

2 COMPARISON OF SOFTWARE DEVELOPMENT PROCESS MODELS

A software process model is the essential representation in the software development and its evolution. The software is being developed in accordance with its process model and its stages. It plays a key role for the delivery of the software in terms of its quality. A software process model is a simplified representation of a software process. Each model represents a process from a specific perspective. The advantages and disadvantages of traditional Software Development Process Models are summarised in Table-1. It is also necessary to compare these aforesaid models to compare on different parameters as described in Table-2. The traditional software

process models have their own advantages and disadvantages. The similar way, the Table-3 summarizes the advantages and disadvantages of the agile methods. There are currently six agile methods which already adopted in the industry. Table-4 is summarizing the main features, key points, and other criteria of agile methods. Table-5 discusses the comparison of traditional and agile software process methods. The outcome is the agile method is flexible.

3 RESEARCH GAP AND EXPERIMENTATION

An increased focus on agile methods has brought more focus on measurement in the field since it among other things indicates the company's ability to adapt to the market and take advantage of opportunities that become available due to changes (Erande & Verma , 2008; Arteta and Giachetti , 2004). The lack of an established definition of agility has caused measurement in ASD to be inconsistent (Abrahamsson, 2009). Instead of adding yet another definition to the research field, this study borrows from the work of Conboy (2009) and his comprehensive definition for agility in ISD: The continual readiness of an ISD method to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value (economy, quality, and simplicity), through its collective components and relationships with its environment. Previous research has highlighted the importance of evaluating how well agile practices support organizational goals and values to ensure non-conflicting purposes (Conboy, 2009). Furthermore, prior studies has called for more research regarding agile goals and values (High smith, 2002), measurement in ASD (Abrahamsson, 2009), management-oriented methods (Dingsøyr et al., 2008), the underlying ideas behind ASD (Dingsøyr et al., 2008) and theory-based approaches in ASD (Dingsøyr et al., 2012). The lack of an established definition of agility in ASD has also been pointed out by previous researchers (Abrahamsson, 2009). In order to address this expressed need in research, this study examines how gaps between strategic priorities and perceptual performance in agile practices can be identified. Specifically, the use of five Scrum practices is evaluated considering seven agile aspects. The seven agile aspects are based on Conboy's (2009) taxonomy of ASD agility: A. Creation of change, B. Proaction in advance of change, C. Reaction to change, D. Learning from change, E. Perceived economy, F. Perceived product quality, G. Perceived simplicity. The evaluated practices are: 1. Backlog grooming, 2. Sprint planning, 3. Daily scrum, 4. Sprint review, 5. Sprint retrospective. According to the originators of Scrum, Schwaber & Sutherland (2013), it is a lightweight framework used by teams to develop and sustain complex products. These agile aspects have been prioritized based on previous research outcomes whereas the performance has been measured by sending out survey questions to scrum team members. The strategic weighting was used as reference against which the performance was compared to identify the gaps between priorities and performance. Two indicators were used for performance; (i) mean and (ii) mode of the scrum team members scoring. Weights and indicators for performance were then compared to identify potential areas of improvement. Denotations and formulas for calculations are presented in table-6 and equations 1-3 respectively. Analysis was conducted in the cases where the difference between the weights (W) and any of the performance indicators (M and T) were greater than one.

Table-6: *Symbolic Notation for Calculation*

Symbol	Descriptions
W	Weight of agile aspect in agile practice
x_i	Score i for scrum team member
M	Mean of scrum team members scoring
T	Mode of scrum team members scoring.
S_M	Difference between weight and mean
S_T	Difference between weight and mode

$$S_M = V - M \quad (1)$$
$$S_T = V - T \quad (2)$$

4 RESULT AND INTERPRETATION

The weighting of agile aspects in agile practices are used as a reference to assess performance and identify gaps, this is presented in table-7. The quantitative result shows that each agile aspect is important or very important in at least two practices or more, which indicates that the measured aspects are considered essential in the organization. Furthermore, by considering the average weighting of each aspect or practice, the results show that economy (E) is the most important agile aspect while backlog grooming (1) and sprint retrospective (5) are the most important practices. The weighting of agile aspects in agile practices are used as a reference to assess performance and identify gaps, this is presented in table-7. The quantitative result shows that each agile aspect is important or very important in at least two practices or more, which indicates that the measured aspects are considered essential in the organization. Furthermore, by considering the average weighting of each aspect or practice, the results show that economy (E) is the most important agile aspect while backlog grooming (1) and sprint retrospective (5) are the most important practices.

Table-7*. Weighting of agile aspect* A---G in practices 1-5.

	A	B	C	D	E	F	G
1	5	4	4	3	5	4	5
2	2	4	4	2	5	4	2
3	1	3	4	2	5	4	4
4	5	4	4	4	2	4	2
5	5	4	4	4	5	5	3

Based on the results and discussion above, the key findings are: The agile aspects are important in the organization, which implicates that concepts that are central to the organization have been measured. There are areas where practices either underperform or over perform in relation to strategic weighting of agile aspects. Quantitative data is useful to indicate potential areas of improvement, while it can be complemented with qualitative data to examine underlying causes for gaps. By summarizing the results in a matrix format, it is possible to visualize potential dependencies between agile practices and dependencies between agile aspects.

5 DEGREE OF AGILITY IN XP AND SCRUM

In this part of the comparative analysis, the degree of agility (DA) is measured in terms of the five features relating to agility characterization: flexibility (FY), speed (SD), leanness (LS),

learning (LG) and responsiveness (RS) that may exist at some specific level or lifecycle phase or as a result of the practices used in the phases of XP and Scrum. If any phase or practice (XP or Scrum) supports a particular agility feature, then 1 point is allocated in that particular cell, otherwise 0; and so on. Tables-9, 10, 11 and Figure-1 demonstrate the results of the agility measurement for both XP and Scrum. The agile methods literature was used as the source of the numerical inputs for the analysis, calculated using the following equation (Qumer and Henderson-Sellers, 2006):

$$DA(\text{Object}) = (1/m)\sum m\ DA(\text{Object, Phase or Practices}) \quad (3)$$

The cell values for XP is shown in Table-9 for the five characteristics of agile both the high level (phases) and low level (practices). This analysis is used both a qualitative and a quantitative approach to evaluate the agile methods at both the process level and the practice level. On the basis of this analysis, we may rank the methods as XP has more agile phases but less agile practices than Scrum (Figure-1). The column values of Tables-9 and 10 are summarized in Table-11, which shows the overall degree of agility for both phases and practices of XP and Scrum. These assessments of the degree of agility permit a separate ranking and visual comparison for both a process-based viewpoint and a practice-based viewpoint (Figure-1). There is, of course, no easy and valid mathematical way of combining these two numbers and rankings. Thus, the decision-maker needs to include their own, often subjective, weightings to any evaluation of the most appropriate agile method to be adopted by their organization or for a particular project.

6 CONCLUSION

The Software life cycle models are the tools that help to manage software life cycle. This paper represents comparison of traditional and agile software process models. These models specify process consistency and improvement. Each model has its own strengths and weaknesses. Some models are more appropriate in certain project circumstance than others. The selection of a software lifecycle model for a project is an important decision. The use of an inappropriate model can be detrimental to project success and software quality. There are two composite approach is used. Part one, we surveyed and compare traditional software development models and agile software models, described their advantages and disadvantages, and discuss the features they inherit. Experimentation carried is on in part two. Based on the results and discussion above, the key findings are: The agile aspects are important in the organization, which implicates that concepts that are central to the organization have been measured. There are areas where practices either underperform or over perform in relation to strategic weighting of agile aspects. In this paper, we have analyzed and compared two agile software development methods XP and Scrum by using agility characterization features. This analysis used both a qualitative and a quantitative approach to assess the agile methods at both the phase level and the practice level. On the basis of this analysis, we may rank the methods as XP has more agile phases but less agile practices than Scrum.

Table-1: *Traditional Software Development Process Models Advantages and Disadvantages*

Model Name	Advantages	Disadvantages
Build and Fix Model	• It provides immediate feedback to developers: This shows immediate signs of progress. • Removes planning / design / documentation overhead.	• Increased time is spent on debugging. • Does not promote documentation and therefore produced software is costlier to maintain. • Design changes cost must more after coding has started.
Waterfall Model	• It enforces a disciplined engineering approach. • Verification is done after each phase. • Documentation can reduce maintenance cost. • Feedback loops help to correct faults immediately.	• It only incorporates iteration indirectly, thus changes may cause considerable confusion as the project progresses. • As the client usually only has a vague idea of exactly what is required from the software product, this model has difficulty accommodating the natural uncertainty that exists at the beginning of the project.

Prototyping Model	• When client is not confident about the developer's capabilities, he asks for a small prototype to be built. Based on this model, he judges capabilities of developer. • It reduces risk of failure, as potential risks can be identified early and mitigation steps can be taken.	• It is a slow process. • Too much involvement of client is not always preferred by the developer. Once we get proper requirements from client after showing prototype model, it may be of no use. That is why; sometimes we refer to the prototype as "Throw-away" prototype.
Incremental Model	• Generates working software quickly and early during the software life cycle. • More flexible - less costly to change scope and requirements. • Easier to test and debug during a smaller iteration. • Easier to manage risk because risky pieces are identified and handled during its iteration.	• At each stage of the model, a new build is coded and then integrated into the structure, which is tested as a whole. • Each phase of an iteration is rigid and do not overlap each other. • Problems may arise pertaining to system architecture because not all requirements are gathered up front for the entire software life cycle.
Spiral Model	• The model uses prototyping as a risk reduction mechanism and allows for the development of prototypes at any stage of the evolutionary development. • It maintains a systematic step-wise approach, like the classic life cycle model, but incorporates it into an iterative framework that more reflect the real world.	• Demands considerable risk-assessment expertise. • It has not been employed as much proven models and hence may prove difficult to 'sell' to the client, especially where a contract is involved, that this model is controllable and efficient.
Iterative Model	• In this model better testing is possible in each iteration. • This model does not require high complexity rate. • In this model feedback is generated quickly.	• It requires planning at technical level. • It is not easily understandable.
X-Model	• Reusability, clear requirements for large systems	• Increases complexity, no risk analysis, cost is high
V-Model	• Well defined requirement specifications • High amount of risk analysis involved at beginning. • Good for critical projects. • Early production, easy to manage due to rigidity of model and easy to understand.	• Not a good model for object-oriented projects. • Not a good model for long and on-going projects. • Not suitable were requirements have high risk of changing. • Costly model, not suitable for small projects. • Long implementation time

***Table-2**: Comparison of Traditional Software Development Process Models*

Parameters	Waterfall	Iterative	Prototype	Incremental	Spiral	Build & Fix
Requirement specifications	Yes	Yes	No	No	No	No
User involvement in phases	Only at requirement analysis	Yes	High	Yes (intermediate)	High	Yes
Flexibility	No	Good	No	Low	Yes	Yes
Overlapping phases	No	Yes	Yes	No	Yes	Yes
Risk Analysis	Only at beginning	Low	Yes	No	Yes	No
Expertise required	High	High	Medium	Medium	High	Medium
Reusability	Limited	Yes	Yes	Yes	Yes	Weak
Simplicity	Simple	Simple	Low	Medium	Medium	Simple
Development time	Long	Medium	Long	Long	Long	Depends upon projects
Incorporation of changes	Difficult	Easy	Easy	Easy	Easy	Difficult
Success rate	Low	High	Good	Good	High	Medium

Cost	High	Low	High	Low	High	Low
Cost control	Yes	No	No	No	Yes	Yes

***Table-3**: Agile Software Development Process Models Advantages and Disadvantages*

Agile method	Overview	Advantages	Disadvantages
Scrum	• An iterative and incremental • framework, feedback- driven, where self- organizing development team works as a unit to reach a common goal • 30-days release cycles	• High communication level with team member • Client participation • Self-organizing team • Respect team members	• lack of documentation • Close interaction • Lack of formal leader • Specialized skills, test plan • Can easily get out-off track
Lean Development	• Optimizing efficiency and minimizing in producing software that does not address the customers' needs	• Waste elimination • No need for defining a complete specification • Deliver high-quality systems quickly • Optimizing delivered business value	• Difficult to develop the system as a single unit • No clear requirements definition can increase implementation time.
Extreme programming (XP)	• Frequent "releases" in short development cycles, customer-driven development • XP argue that the code is the only truly important product of the system development • process daily builds	• Team communication, • Technical practice • Frequent feedback • Encourage for effective actions • Respect team members • Open to employee creativity, freedom, self-control and innovation	• Implicit design • Hard to write good test • Frequent iterations compromise quality • better outcome if all team members can work at the same location
Adaptive Software Development (ASD)	Adaptive practice focuses on collaboration and learning as a technique to build complex systems, Repeating series of speculate, collaborate, and learn cycles.	• Extensive users involvement • Requirements evolution • High quality software • Early delivery.	• Extensive testing increases the cost • Projects become bigger than they were visualized in the initial stages.
Agile Modeling	The agile version of Model-Driven Development	• Continuous attention to technical excellence and good design. • Rapid, continuous delivery of software. • Customers, developers and testers interaction • Welcoming late changes in requirements	• Lack of documentation • Final outcome is not clear • Fit only senior programmers
Crystal Methods	Collection of agile development methodologies focuses on six primary aspects: people, interaction, community, communication, skills, and talents.	• Human-powered • Adaptive • Light-weight methodology	• Critical decisions regarding the architecture of the application are made by individuals and not by the entire team.
Dynamic System Development Methodology (DSDM)	Its origins the Rapid Application Development (RAD) Methodology.	• Requirements continue to evolve • Adheres strictly to agreed timelines and budget • Stakeholders involvement • Strong emphasis on system testing	• documentation are complex and time consuming can reduce the robustness of code • Skilled developers / development teams are e required

Feature-Driven Development	Oriented on resulting out small blocks of client valued functionality.	• Multiple teams working in parallel • Tiny delivered software • Map directly onto an object domain model • Coded directly • Assembled in component sets	• Not as powerful on smaller projects • Lack of documentation. • Reliance on programmer. • individual code ownership

Table-4: *Comparison of Agile Software Development Process Models*

Software	XP	Scrum	FDD	ASD	DSDM	Crystal
Development Process	Short releases	Scrum teams	Domain object	The project mission	Active user involvement	Staging
	Metaphor	Product backlog	Development by feature	Development by component	Empowered teams	Holistic delivery
	Simple	Sprint	Individual class	Collaborative teams	Frequent product delivery	Parallelism and flux
	Testing	Sprint review	Feature items	Joint Application Development	Fitness for business purpose	User viewings
	Refactoring	Not Known	Inspection	Customer group reviews	Iterative and incremental	Revision
	Pair programming	Not available	Regular builds	Software inspection	Reversible changes	Not available
	Collaborative ownership	Not available	Not available	Not available	Requirements are baselined at high-level	Not available
	Continuous integration	Not available	Not available	Not available	Integrated testing	Not available
	On-site customer	Not available	Not available	Not available	Collaboration and cooperation among stakeholders	Not available
	Coding standard	Common coding standard	compliance to improve the readability of code	As prescribed by PMI	Not known	Not known
Project management process	The planning game	Scrum Master	Reporting/ visibility of results	Adaptive cycle Planning	Not specified	Monitoring of a progress
	Not known	Sprint / Scrum meeting	Not known	Adaptive management model	Not known	Not known
Software configuration control process / support process	Not specified	Not specified	Configuration management	Not specified	Not specified	Not specified
Process management process	Not specified	Not specified	Not specified	Project Postmortem	Not specified	Reflection workshops methodology tuning

Table-5: *Comparison of Traditional and Agile Software Development Process Methods*

Characteristics	Traditional project management	Agile project management
Organizational structure	Linear	Iterative
Project scale	Large scale	Small and medium scale
Development model	Life cycle model	Evolutionary delivery model
User requirements	Clearly define coding or implementation	Iterative input
Client involvement	Low	High
Restart cost	High	Low
Development process	Life cycle model	Low
Development model	Fixed	Easily changeable
Testing	Once coding is done	Early iteration

Architecture	Creates current and predictable requirements	Creates current requirements
Requirements	Standard and known in advance	Emergent with rapid changes
Adaptability to change	Change sustainability	Change adaptability
Development approach	Predictive	Additive
Development orientation	Process-orientation	People-orientation
Planning scale	Long-term	Short-term
Management style	Command and control	Leadership and collaboration
Learning	Continuous learning while developing	Learning is secondary while developing
Documentation	High	Low

Table-8: *Descriptive statistics for daily scrum*

Factor	Creation	Proaction	Reaction	Learning	Economy	Product quality	Simplicity
Weighting (V)	1	3	4	2	5	4	4
Mode (T)	2	4	3	1	4	3	5
Mean (M)	2,81	3,25	2,94	1,81	3,44	2,94	4,13
Standard Deviation	1,17	1,44	1,29	0,91	1,31	0,85	1,02
Interval	[1, 5]	[1, 5]	[1, 5]	[1, 3]	[1, 5]	[1, 4]	[2, 5]
Difference (S_T)	-1	-1	1	1	1	1	-1
Difference (S_M)	-1,81	-0,25	1,06	0,19	1,56	1,06	-0,13

Table-9*. Degree of agility in XP*

	Agility Features					
Agile Method: XP	FY	SD	LS	LG	RS	Total
(i) Phases						
Exploration	1	1	0	1	1	4
Planning	1	1	0	1	1	4
Iteration to release	1	1	0	1	1	4
Productionizing	1	1	1	1	1	5
Maintenance	1	0	0	1	1	3
Death	0	1	0	0	0	1
Total	5	5	1	5	5	21
Degree of Agility	5/6	5/6	1/6	5/6	5/6	21/(6*5)
(ii) Practices						
The Planning Game	1	1	0	1	1	4
Short Release	1	1		1	1	4
Metaphor	0	1	1	0	0	2
Simple Design	1	1	1	1	1	5
Testing	1	1	0	1	1	4
Refactoring	1	1	1	1	1	5
Pair Programming	1	0	0	1	1	3
Collective Ownership	1	0	0	1	1	3
Continuous Integration	1	1	1	1	1	5
40-Hour Week	0	0	0	1	0	1
On-site Customer	1	0	0	1	1	3
Coding Standards	1	1	1	1	1	5
Total	10	8	5	11	10	44
Degree of Agility	10/12	8/12	5/12	11/12	10/12	44/(12*5)

Table-10. *Degree of agility in Scrum*

	Agility Features					
Agile Method: Scrum	FY	SD	LS	LG	RS	Total
(i) Phases						
Pre-Game	1	1	0	1	1	4
Development	1	1	0	1	1	4
Post-Game	0	1	0	0	0	1
Total	2	3	0	2	2	9
Degree of Agility	2/3	3/3	0/3	2/3	2/3	9/(3*5)
(ii) Practices						
Scrum Master	1	1	0	1	1	4
Scrum Teams	1	1	0	1	1	4
Product Backlog	1	1	0	1	1	4
Sprint	1	1	0	1	1	4
Sprint Planning Meeting	1	1	0	1	1	4
Daily Scrum Meeting	1	1	0	1	1	4
Sprint Review	1	1	0	1	1	4
Total	7	7	0	7	7	28
Degree of Agility	7/7	7/7	0/7	7/7	7/7	28 / (7*5)

Table-11. *Degree of Agility in XP and Scrum*

Process & Practices	XP	Scrum
Phases	19/30 = 0.63	10/15 = 0.67
Rank	2	1
Practices	42/60 =0.70	30/35 =0.86
Rank	2	1

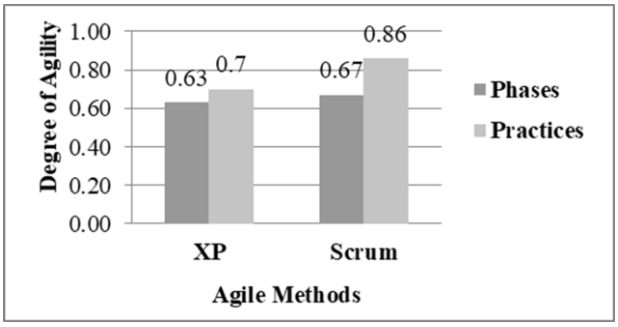

Fig. 1. *Degree of agility – a comparison of Phases and Practices for XP and Scrum.*

REFERENCES

[1] Murat güneştaş, “A study on component based software engineering”, a Master’s thesis in Computer Engineering, Atılım University, JANUARY 2005

[2] Ivica Crnkovic, Stig Larsson; Michel Chaudron, “Component-based Development Process and Component Lifecycle.” Online Available: http://www.mrtc.mdh.se/publi cations/0953.pdf

[3] Luiz Fernando Capretz Y, “A New Component-based software life cycle model”, Journal of Computer Science 1 (1): 76-82, 2005, ISSN 1549-3636, Science Publications, 2005

[4] K.Kaur, H Singh, “Candidate process models for component based software development”, Journal of Software Engineering 4 (1):16-29, Academic Journal Inc, India 2010.

[5] Sajid Riaz, “Moving Towards Component Based Software Engineering in Train Control Applications”, Final thesis, Linköpings universitet, sweden, 2012

[6] Ian Somerville, Software Engineering, Addison Wesley, 9th ed, 2010.

[7] Nabil Mohammed Ali Munassar, A.Govardhan, “A Comparison between Five Models of Software Engineering”, International Journal of Computer Science Issues, Vol.: 7, Issue: 5, Sep, 2010.
[8] Craig Layman and Victor Basili, “Iterative and Incremental Development: A Brief History”, IEEE Computer 2003.
[9] W. Royce, “Managing the Development of Large Software Systems”, presented at the Proceedings of IEEE WESCON, 1970.
[10] Rajesh Shah, ”A Comparative Study of two Software development Approach Traditional and object Oriented”, International Journal of Advanced Research in Computer Science and Software Engineering”, Vol -5 Issue-5, pp 1383-1387, ISSN-2277128X, 2011.
[11] Barry Boehm, "Spiral Development: Experience, Principles, and Refinements", edited by Wilfred J. Hansen, 2000.
[12] Jintao zeng, Jinzhong Li, Xiaohui Zeng, Wenlang Luo, “A Prototype System of Software Reliability Prediction and Estimation”, IITSI 2010.
[13] Craig Larman and Victor Basili, “Iterative and Incremental Development: A Brief History”, IEEE Computer, June 2003.
[14] C. Melissa McClendon, Larry Ragout, Gerri Akers, “The Analysis and Prototyping of Effective Graphical User Interfaces”, October 1996.
[15] Conboy K, “Agility From First Principles: Reconstructing the Concept of Agility in Information Systems Development”, Information Systems Research, 2009
[16] Dingsøyr T, Nerur S, Balijepally V, and Moe N.B, “A decade of agile methodologies: Towards explaining agile software development”, Journal of Systems and Software, vol. 85, no. 6, pp. 1213-1221, 2012
[17] Abrahamsson P, Conboy K, Wang X, “Lots done, more to do’: the current state of agile systems development research”, European Journal of Information Systems, 18, pp. 281–284, 2009
[18] Dingsøyr, T., Dybå, T., and Abrahamsson, P, “A preliminary roadmap for empirical research on agile software development”, In Agile, 2008. AGILE'08. Conference, pp. 83-94, IEEE, 2008
[19] Erande, A. S., & Verma, A. K, “Measuring agility of organizations–a comprehensive agility measurement tool (CAMT)”, International Journal of Applied Management and Technology, 6(3), 3, 2008
[20] Guion, R. M, “Performance measurement and theory”, Hills-dale, NJ: Erlbaum, pp 267-275, 1983
[21] Highsmith J, “Agile Software Development Ecosystems”, Addison-Wesley, Boston, 2002
[22] Neely A, “Business Performance Measurement: Theory and Practice”, Cambridge University Press, 2002
[23] Schwaber K, Sutherland J, “The Scrum guide – the definite guide to Scrum: the rules of the game:, Scrum.org, 2013
[24] Qumer A and Henderson-Sellers B, “Measuring agility and adoptability of agile methods: A 4-Dimensional Analytical Tool”, Procs. IADIS International Conference Applied Computing 2006, IADIS Press, 503-507.
[25] Williams L, Krebs W, Layman L, Anton A.I and Abrahamsson P, “Toward a framework for evaluating Extreme Programming”, Empirical Assessment in Software Engineering, Edinburgh, pp 11-20, 2004
[26] Kitchenham B.A and Jones L, “Evaluating software engineering methods and tool part 5: the influence of human factors”, ACM SIGSOFT Software Engineering Notes, 13-15, 1997
[27] Agile Manifesto, “Manifesto for Agile Software Development”, 2001
[28] Beck K, “Extreme Programming Explained”, Addison-Wesley Pearson Education, Boston, 2000
[29]] Boehm B and Turner R, “Balancing agility and discipline: evaluating and integrating agile and plan-driven methods”, Proceedings of the 26th International Conference on Software EngineeringIEEE Computer Society, Washington, DC, USA, pp 718-719, 2004
[30] Kitchenham B.A and Jones L, “Evaluating software engineering methods and tool part 5: the influence of human factors”, ACM SIGSOFT Software Engineering Notes, pp 13-15, 1997
[31] Koch A.S, “Agile Software Development: Evaluating the Methods for Your Organization”, Artech House, Inc, London, 2005
[32] Palmer S.R and Felsing J.M, “A Practical Guide to Feature-Driven Development”, Prentice-Hall Inc, Upper Saddle River, 2002