Towards A Framework for Understanding the Relationships between Classical Software Engineering and Agile Methodologies

Li Jiang
The University of Adelaide, Australia
School of Computer Science, The University of Adelaide, SA, 5000, Australia
+61 8 8303 6191
ljiang@cs.adelaide.edu.au

Armin Eberlein
University of Sharjah, UAE
Computer Engineering Department, American University of Sharjah, UAE
+971 6 515-2936
eberlein@ucalgary.ca

ABSTRACT

There is an ongoing debate in the software engineering (SE) community over the usefulness and applicability of classical SE methodologies versus agile methodologies. Based on an investigation of the philosophical origins, the history and the technological support of representative classical SE methodologies and agile methodologies, a framework is proposed in this paper to help understand the relationship between these different approaches. The framework proposed provides a novel, five-dimensional ways in which to consider the concepts, historical and technological background of the methodologies, the characteristic differences between them, the variety of skills, and the economic, technological and organisational conditions needed to execute them. The framework has been formed by combining five techniques of research analysis: **C**ontextual; **H**istorical; **A**nalysis by analogy; **P**henomenological; and **L**inguistic. This framework (**CHAPL**) helps software engineers understand the nature of SE methodologies more objectively and at a fundamental level in order to select the best practices and suitable SE methodologies for a software project.

Categories and Subject Descriptors

D.2.1 [Requirements/Specifications]: Software Engineering Methodologies
D.2.9 [Management]: Software Process Models, Software Process Maturity Assessment Model, Software Life Cycle.

General Terms: Management, Standardization,

Key words

Software engineering, software engineering methodology, software process, waterfall model, agile methodology, philosophy

1 Introduction

Since the late 1960s, numerous methodologies have been developed to address the various challenges that occur during software development. The advent of 'agile methodologies' as another answer to some software engineering (SE) problems has caused a heated debate amongst software developers since early 2000, with proponents of classical SE methodologies, such as the 'Waterfall' or 'Spiral'[1] model questioning the value of agile methodologies, such as XP and SCRUM[2]. On one hand, software engineers such as Rakitin, and Harrison [1; 2] dismiss agile methodologies and strongly advocate the value of classical SE practice, while others [3; 4] insist that agile methodologies will replace Waterfall-like models and apply to all software projects. This heated debate is sometimes referred to as the "Methodology War" [5; 6].

It appears that the typical characteristics of the debate are that the proponents of the conflicting methodologies:

- describe each other in extreme and biased terms
- devalue the opponents' methodologies and/or practices
- justify their own values through either experience-based explanation or inadequate comparisons between the methodologies.

We believe that this war is detrimental to SE practices. In order to end the Methodology War, some researchers have presented the similarities and compatibilities between the two methodologies [5; 7-10]. Furthermore, Boehm [11; 12] argues that each methodology offers its own values to different software

[1] Different terms are used in the software engineering domain that mean essentially the same thing in different contexts, such as software engineering method, software engineering methodology, software process model, software engineering techniques. In this paper, the term 'classical software engineering methodology' is used to represent all terms mentioned above and refers to the waterfall, spiral model, CMM [16] and the like where upfront requirements engineering consists of documentation and detailed plans. However, it is acknowledged that differences between these terms in some contexts have also been discussed by researchers such as [17, 18]. Notice that we use 'agile methodology' (see footnote 2) while 'agile method' is also widely used.

[2] Agile methodologies refer to a collection of agile methods, predominately XP and SCRUM and the like, where the essential 12 agile principles are followed as described in [4, 19].

projects and he provides suggestions and arguments to guide those wishing to use agile methodologies in an organisation where classical SE methodologies are also being used.

Glass [5] argued that the disagreement between advocates of agile and classical SE methodologies is largely based on conflicting beliefs about how a SE process is best organised and performed, which indicates that the dispute may in reality be a problem of individual or organisational attitudes, work habits and misconceptions about the different methodologies and their relationships to one another. Based on our research of literature and consultation with industrial software engineers [13-15], we found that most of the questions raised in the methodology debate are fundamental disagreements about how to approach, structure, and manage a SE process. We are convinced that one of the essential reasons for the debate is that the principles and practices of the two methodologies lack a framework that helps understand the nature of SE methodologies. It appears that no attempt has yet been made to comprehensively analyse the compatibility and complementarity of the methodologies. We believe that it is necessary for the SE community to look at this issue earnestly, and recognise the interrelationships between the two methodologies in order to advance SE research and practices.

Based on an investigation of the origin, history and technological support of classical and agile SE methodologies, we propose a framework which helps to understand the affinity of the methodologies for one another and the complementary and combinative relationship between the methodologies. This framework provides five analytical views for considering the concepts, the historical and technological backgrounds of the methodologies, the characteristic differences between them, the variety of skills, and the economic, technological and organisational conditions needed to execute them. The five views are formed from five fundamental philosophy analysis techniques: (1) ***C***ontextual analysis; (2) ***H***istorical analysis; (3) ***A***nalysis by analogy; (4) ***P***henomenological analysis; (5) ***L***inguistic analysis (**CHAPL**). Using our proposed CHAPL framework, we argue that all existing methodologies include valuable engineering practices and knowledge accumulated during the past 40 years. This framework is the first component o the project 'Exploring the Origins and Complementarities of *C*lassical and *A*gile *S*oftware *E*ngineering *M*ethodologies Using Qualitative Evidence' (CASEM) [3] [20]. The framework helps software engineers to understand the nature of software methodology more objectively, and select the best practices and suitable SE methodologies for a software project at hand. The results presented in this paper show the current status of our on-going research.

The rest of the paper is organised as follows: Section 2 presents related work. The framework is presented in Section 3. Conclusions and discussion of future work are summarised in Section 4.

2 Related Works

One of the key debates between proponents of classical SE methodologies and supporters of agile methodologies is the value of 'heavy processes' which are advocated by the Waterfall and other plan-based process models supported by the philosophy underlying the CMM[4] model. To mitigate this debate, Reifer argues that XP and the CMM are philosophically compatible [21]. He contends that further research is needed to provide guidelines for utilising the best practices from both XP and the CMM. In the same vein, Boehm [11] notes that pure agile methodologies or pure classical discipline alone cannot provide all solutions required for projects and a mixture of the best practices from both classical SE and agile methodologies is highly desirable [11]. After examining the merits of the XP process, Paulk [22] argues that blind rejection of any of the practices in either agile or classical methodologies is detrimental to software engineering practices. Glass [5] also advocates borrowing ideas from both methodologies to help in software engineering. Similarly, based on comparisons between the practices in CMMI and agile methodologies, Turner and Jain [7] show that agile practices support 11 components of CMMI, which underscores the points that CMMI and agile are complementary. Despite this work, our research has shown that current literature still does not interpret well enough the relationship between the two categories of methodologies in terms of their fundamental operating philosophies. The rest of the paper will discuss in greater detail the framework to tackle this issue.

3 A Unified Framework For Understanding SE Methodologies

3.1 Overview of the Framework

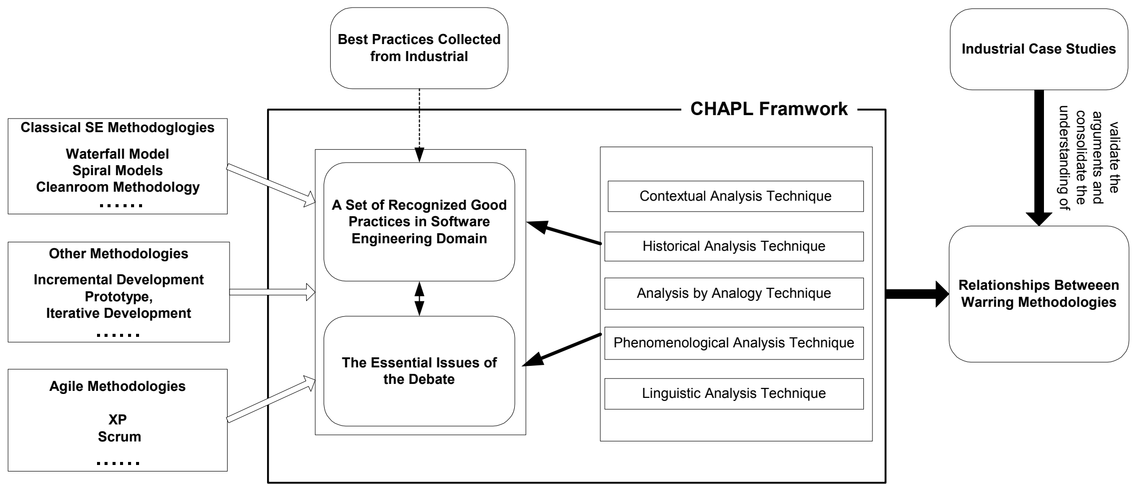
As the Methodology debate is essentially a dispute about fundamental operating philosophies or styles of operation in a software development process, it is reasonable to explore not only the fundamental differences between methodologies but also their similarities. Therefore, the hypothesis of this research is:

Classical and agile methodologies have common philosophical origins and are technically compatible and complementary

The exploration of the relationships between the concerned methodologies has been conducted during the current research using the CHAPL framework as illustrated in Fig. 1. The inputs of the framework are best practices of classical SE methodologies, agile methodologies as well as best practices from industry. Table 1 shows key best practices of SE methodologies [23, 24]. The output of the framework is the relationships between various methodologies. The hypothetical relationships extracted from the theoretical analysis will be consolidated using industrial case studies in which these practices were actually applied.

[3] The project, 'Exploring the Origins and Complementarities of *C*lassical and *A*gile *S*oftware *E*ngineering *M*ethodologies Using Qualitative Evidence' (CASEM), is a part of the project called "A Knowledge Based Approach For Software Engineering Methodology Selection Based on Project Characteristics (ASEMS). The aim of ASEMS is to develop a framework to help select the best practices based on characteristics of software project, compatibility and technical combinability of methodologies.

[4] CMM (Capability Maturity Model), and CMMI (Capability Maturity Model Integration) are assessment models used to assess the maturity of a company or processes. It advocates the same values such as well-planned, defined, organized, and engineered software processes in software development lifecycle with the classical software process models, represented by the Waterfall-liked model. From the philosophy perspective, we classified them in the same category.


Best Practices Collected from Industrial
Industrial Case Studies
CHAPL Framwork
Classical SE Methodoglogies
Waterfall Model
Spiral Models
Cleanroom Methodology
......
Other Methodologies
Incremental Development
Prototype,
Iterative Development
......
Agile Methodologies
XP
Scrum
......
A Set of Recognized Good Practices in Software Engineering Domain
The Essential Issues of the Debate
Contextual Analysis Technique
Historical Analysis Technique
Analysis by Analogy Technique
Phenomenological Analysis Technique
Linguistic Analysis Technique
validate the arguments and consolidate the understanding of
Relationships Betweeen Warring Methodologies

Legents:

1 The symbol "⇨" represents extraction of the fundamental practices in the methodologies as described in the rectangles shown above as the foci of analysis

2 The symbol "➡" represents the adequate explanation, arguments, analysis and inferences of the relationship between the the practices in the methodologies

3 The symbol "⇢" represents the best practices collected from industry or industrial case studies to understand the nature of the practice and validate the hypothesis

4 The symbol "⬇" represents summarization of the finding in the research to consolidate the hypothetical relationships between the warring methodologies through theoretical an alysis and empirical data collected from industry

5 The symbol "↕" represents the recoginized good practices are related to the essential issues of the debate.

Figure 1 A Framework For Analysis of The Relationships Between Methodoloies in Software Engineering

The best practices shown in Table 1 have been analyzed using the following five analysis techniques: ***C***ontextual Analysis; ***H***istorical Analysis; ***A***nalysis by analogy; ***P***henomenological Analysis; and ***L***inguistic Analysis. CHAPL helped software engineers to rationally analyze the warring issues and to derive logically the applicability of a software process model to a particular software project. The essential issues of the debate, i.e., those aspects that are at the core of the heated debate in the methodology dispute, are also included in the framework. We believe that these issues can serve as a bridge that links the methodologies if they are analysed logically and objectively using CHAPL.

The CHAPL framework is still in its early stage of research. We believe that this framework will help to understand the relationship between the methodologies and select rationally best practices and suitable SE methodologies for a software project.

3.2 Five Analysis Techniques

The five techniques used in our framework are the most often used techniques in philosophical analysis. These techniques have proven useful when explaining phenomena in systems engineering [25] and when discussing the philosophy of technology [26]. In this research, CHAPL forms an essential part of the research framework and it is used to analyse and explain the fundamental practices of classical SE methodologies and agile methodologies. The framework and the basic idea of each analysis technique and how each was used during the current research are briefly summarised as follows:

(1) **Contextual analysis.** Using contextual analysis, the first activity was to identify the relationships between the practices, the methodologies, and the characteristics of software projects. These relationships might include, the 'is a' relationship; 'is part of' relationship; 'the affinity of one methodology for another'; 'complementarity'; 'comparability; and 'conflict'. The second activity was to identify the economic, technological and organisational conditions for using each of the methodologies, such as the training cost, complexity, and the application cost of the practices in the two methodologies. Other issues, including decision making for selection of practices and the impact of organisational learning and change of the use of methodologies were also analysed.

(2) **Historical analysis**. Using historical analysis, we focussed on understanding a methodology and its practices by studying its history and its objectives. In this way, we were able to determine if and how it is being used, how it might have changed or how it has evolved into contemporary practices which might be simple, less sophisticated or more complicated. This type of analysis is very important since we cannot arrive at a rational understanding and conclusion about the origin of methodologies unless we understand how methodologies evolve in response to changes of social and technological contexts.

(3) **Analysis by analogy**. With this technique, we focussed on the discussion of individual practices in each of the methodologies and how they are related to the practices in other methodologies. We also explored how the comparison could help better understand the commonalities between the methodologies in terms of organisational or management philosophies about software processes, and patterns of individual or group behaviour during a software engineering project.

(4) **Phenomenological analysis**. Phenomenological analysis focuses on understanding the experience of software engineers and how software engineers themselves make sense of their experiences of using different software

Table 1 Best SE Practices Addressed By Three Methodologies

Best SE Practices		**Waterfall** [27]		**Spiral** [28]		**XP**[29] **and SCRUM** [30]	
		Contains	**Implicitly Mentioned**	**Contains**	**Implicitly Mentioned**	**Contains**	**Implicitly Mentioned**
Project planning and management practices	Automated estimation tools	×	×	√	×	×	×
	Evolutionary delivery	×	×	√	×	√	×
	Measurement	√	×	√	×	×	<
	Productivity environments	×	<	√	×	√	×
	Risk management planning	×	<	×	×	×	√
	Track defects against quality targets	×	×	×	×	×	×
	Treat people as the most important resource	×	<	×	<	√	×
	Release planning and management	×	<	<	×	×	<
Requirements engineering practices	Manage and trace requirements (Change board)	×	√	×	√	×	<
	Throwaway user interface prototyping	×	×	√	×	×	√
	JAD sessions	×	<	×	<	×	<
Design practices	Information hiding	×	×	×	×	×	×
	Design for change	√	×	×	√	×	×
	Use system-based software design	×	<	×	√	×	×
	Ensure data and database interoperability	×	√	×	√	×	<
	Define and control interfaces	×	<	×	×	×	√
	Design twice, code once	×	×	×	×	×	×
	Assess reuse risks and costs	×	×	√	×	×	×
Construction or Implementation practices	Source code control (Configuration management)	×	×	√	√	√	√
	Incremental development and integration	×	×	×	√	√	×
Software Testing and Quality assurance practices	Manage testing as a continuous process	×	√	×	√	√	×
	Compile and smoke test frequently	×	×	×	√	√	×
	Branch-coverage testing	√	×	√	×	×	×
	Inspections (of requirements and design)	√	×	√	×	×	×
Process improvement	Software Engineering Institute's Software Capability Maturity Model	×	<	×	<	×	<
	Software Engineering Process Groups	×	<	×	<	×	<

Notes: "√" indicates that the methodology contains the good practice either explicitly or implicitly

"×" indicates that the methodology does not explicitly mention the corresponding good practice

"<" indicates that the methodology includes partially the good practice either explicitly or implicitly

practices. Special focus was on the ways in which different practices are included in a methodology and how the individual practices relate to one another. We were also concerned with understanding the characteristic differences between practices and the variety of skills and technologies needed to perform them. Such analysis involved interdisciplinary and multidisciplinary consideration of what is right and good (practices and ethics), knowledge (epistemology), and the structure of methodology (metaphysics) within the given context of a software project.

(5) **Linguistic analysis**. Linguistic analysis helped to clarify and correct both practical and theoretical usage of terms. The terms and their connotations and the concepts underlying the practices included in the methodologies were analysed, classified, and compared so that common interpretation and understanding could be achieved.

It should be mentioned that from a philosophical perspective, there was no clear boundary between the five analytical techniques discussed above [26]. Thus, a mixture of techniques was used in order to provide the appropriate queries, answers and detailed analysis of the key issues which were the focus of this research. To use each technique a set of fundamental questions was developed. These questions were used to help explore answers to the essential issues in the debate. It was felt that by examining the fundamental practices in classical SE methodologies and agile methodologies against the set of questions, the origin and interrelationships of the two methodologies in question were likely to be identified.

As an example, Table 2 lists a set of key questions that were developed in this research and used in historical analysis and contextual analysis of classical SE and agile methodologies. The essential issues of the debate shown in the table were formed from analysis of the literature. The answers to these questions help understand the relationships between the debated methodologies.

It is worth mentioning that questions in the framework were not meant to cover all the philosophical issues raised in the debate. Some questions were more general and aimed at establishing the context in which the methodologies were being applied. For example, the 'typical project background' shown in the table (see QC 3) is characterised by a set of widely used project attributes, such as project type, size of software project, or requirement volatility (refer to [18] for more information). Similarly, the question relating to 'economic, technological and organisational conditions for using a methodology' (see QC 4) was also very broad. Within the research, this question referred to economic constraints, technological constraints, market

Table 2. Examples of the Questions Proposed in the Framework

Examples of the Analysis Dimensions	Questions for Examining the 'Warring' Methodologies	Essential Issues of the Debate To Be Answered by the Framework
Historical Analysis	**Qquestions related to the methodology as a whole:** QH 1: What are the major purposes for creating the methodology? QH 2: What is the social background based on which the methodology was created? QH 3: What is the technological background on which the methodology was created? QH 4: How complex is the methodology? QH 5: How expensive is the methodology (including training cost and application cost)? **The following questions focus on essential practices in each methodology:** QH 6: What is the purpose for using the practice? QH 7: What is the social background for using the practice? QH 8: What is the technological background for using the practice? QH 9: Can the practice be used independently in any specific methodology? QH 10: What practices are used in exactly the same way in other methodologies? QH 11: What practices are used to achieve the same objectives in other methodologies?	● Requirements acquisition. ● Roles and values of documentation (including all documentation in software process) ● Requirements change management ● Software release ● Productivity measure (how to measure the progress) ● Role and value of software processes ● Role and ways of developing a software plan ● Values of developers ● Communication within the team ● Team organisation ● Software testing ● Measurement of customer satisfaction ● Software quality and its measurement ● Applicability of a methodology to different software projects
Contextual Analysis	QC 1: What are the relationships between the practices within the methodology? QC 2: What are the relationships between the two methodologies? QC 3: What is the typical project background (characterised by a set of widely used project attributes) for using each practice? QC 4: What are the economic, technological and organisational conditions for using a particular practice and methodology? QC 5: How expensive is the practice (including training cost and application cost)? QC 6: How complex is the practice? QC 7: What are the decision criteria for the selection of one practice and/or methodology over another?	

Note: "QH" stands for questions related to history analysis. "QC" stands for questions related to contextual analysis.

constraints, culture and developers, and the project manager's educational background in the organisation where the methodologies were used.

Based on the theoretical exploration of classical SE and agile methodologies and the recounting of practical experiences of using these methodologies, the answers to the questions listed in Table 1 will provide sufficient arguments to support the hypothesis that there are common philosophical origins and intercommunity relationships between classical software engineering and agile methodologies.

4. Conclusion and Future Work

As with any technology or tool invented by human beings, all SE methodologies have limitations [31]. Discussions involving these methodologies would be better used in forming an understanding of their advantages and limitations, and how to use them, tailor them, and improve them in SE processes rather than in trying to prove that one methodology is universally superior to the other. Thus, we should stand back and look at the origin of these methodologies, the philosophies behind the methodologies, and the technologies and social environment which supported the generation of these methodologies. Due to the fact that there is no model that allows us to reason about the suitability of SE methodologies [31] in any particular circumstance, efforts should be made to develop a reasoning mechanism that assists in the selection of SE methodologies. The development of such a mechanism is part of the larger research project of which the discussion presented here is a part. This framework serves as the foundation for building the reasoning mechanism discussed above.

It has to be mentioned that the framework presented in this paper is still part of on-going research and is still subject to further refinement. Future research will focus on more specific questions and provide complete answers to each question raised in the framework. Moreover, we will also conduct a survey to elicit critical information about the ways in which industry tailors software practices and methodologies, and the compatibility and effectiveness of combining software methodologies. Furthermore, based on the answers to the questions and the survey results, some quantitative and qualitative data are likely to be obtained to confirm the common origin of, and the intercommunity between, classical SE and agile methodologies. For example, the number of practices that were used in the same way to achieve the same objective in different methodologies and software project is to be investigated in order to provide sufficient information to confirm the hypothesis of our research.

References

[1] Rakitin S. 2001. "Manifesto Elicits Cynicism," Computer, Dec. 2001, p. 4.

[2] Harrison, W. 2003. Is Software Engineering As We Know It Over the Hill?, IEEE Software, Vol. 20, Issue: 3. pp: 5- 7,

[3] Nerur S. and Balijepally V.G. 2007. "Theoretical Reflections on Agile Development Methodologies," Communications of the ACM, Vol. 50, 2007, pp: 79-83

[4] Beck K., *et. al*. 2001. Manifesto for Agile Software Development, http://agilemanifesto.org/principles.html, 2001 (As shown in the webpage)

[5] Glass, R.L. 2001. Agile Versus Traditional: Make Love Not War. Cutter IT Journal, Vol. 14, No. 12, 2001, pp: 12-18.

[6] Austin, R.; Devin, L. 2003. Beyond Requirements: Software Making As Art, IEEE Software, Vol. 20, Issue 1, pp:93 - 95

[7] Turner R. & Jain A. 2002. Agile Meets CMMI: Culture Clash or Common Cause? In Wells D. and Williams L. (Eds.): LNCS , XP/Agile Universe 2002, pp. 153–165.

[8] Fritzsche M., Keil P. 2007. Agile Methods and CMMI: Compatibility or Conflict? e-Informatica Software Engineering Journal, Vol. 1, Issue 1, 2007

[9] Paetsch F., Eberlein A. and Maurer F. 2003. Requirements Engineering and Agile Software Development, Proceedings of the IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE-2003), Linz, Austria, 2003, pp. 308 -313

[10] Eberlein A. 2003. Requirements Engineering and Agile Methods: Can they benefit from each other? Position Statement in the Proceedings of the Canadian Invited Workshop on Scaling XP/Agile Methods, Banff, Canada

[11] Boehm, B.W. 2002. Get Ready for Agile Methods, with Care. IEEE Computer, 2002.

[12] Boehm B. W., Turner R, 2003. Balancing Agility and Discipline. Addison-Wesley.

[13] Bound D. *et al*. 2007. Delivering IT Solutions in the Real World! Fujitsu Inc. Australia. http://www.cs.adelaide.edu.au/~ljiang/Courses_Information.html (click SE In Industry and then click Resources, then select Fujitsu)

[14] Owen C. *et. al*. 2007. IT Solution Delivery, Adelaide Bank, http://www.cs.adelaide.edu.au/~ljiang/Courses_Information.html (click SE In Industry and then click Resources, then select Adelaide Bank),

[15] Cumpston B. *et. al*. 2007. Ebor Computing, Ebor Computing Inc.http://www.cs.adelaide.edu.au/~ljiang/Courses_Information.html, 2007 (click SE In Industry and then click Resources, then select Ebor Computing)

[16] Paulk M. C. *et al*. 1995. The Capability Maturity Model: Guidelines for Improving the Software Process, Addison-Wesley, Reading, Mass., 1995.

[17] Floyd C. 1986. A Comparative Evaluation of System Development Methods, In Olle, T.W., Sol, H.S., and Verrijn-Stuart, A.A.,(ed) (North-Holland, Amsterdam), 1986, pp:19-54

[18] Jiang L, 2005. A framework for requirements engineering process development. University of Calgary, PhD Thesis, Sept. 2005

[19] Cockburn A. 2002, Agile Software Development, Addison-Wesley, 2002.

[20] Jiang, L. 2006. Exploring Philosophical Origin and Foundation of the Fundamental Practices In Classical Software Engineering Methods and Agile Method with Qualitative Evidence (POSEA), http://www.cs.adelaide.edu.au/~ljiang/ResearchHomePage.html.

[21] Reifer, D.J., 2003. XP and the CMM, IEEE Software, Vol. 20, Issue 3, May-June 2003, pp: 14 - 15

[22] Paulk, M. C. 2001. Extreme Programming from a CMM Perspective. IEEE Software, 2001.

[23] McConnell, S. 2002. IEEE Software, Vol: 19, Issue: 1, pp: 3-5 Jan/Feb. 2002

[24] Software Program Managers Network (SPMN). 1999. 16 Critical Software Practices™ for Performance-based Management. http://www.spmn.com/critical_software_practices.html,1999

[25] Dahlbom B. Mathiassen L. 1992. Systems Development Philosophy, ACM SIGCAS, Computers and Society, Vol. 22, Issue 1-4, Oct. 1992, pp: 12 – 23

[26] Carl M. 1998. The Importance of Philosophy to Engineering, Teorema XVII (3), 1998, pp. 27-47.

[27] Royce W. W. 1970. Managing the Development of large Software Systems: Concepts and Techniques, Proceeding of the IEEE WESTCON, Los Angeles, California, 1970

[28] Boehm B. W. 1988. A Spiral Model of Software Development and Enhancement. IEEE Computer 21(5): 1988, pp: 61-72.

[29] Beck K. and Andres C. 2005. Extreme Programming Explained: Embrace Change. Addison-Wesley, Boston, 2005.

[30] Schwaber K. , Beedle M. 2001. Agile Software Development with SCRUM, Prentice Hall; 1st edition, 2001

[31] Basili V. R. 1996. The Role of Experimentation In Software Engineering: Past, Current, and Future, Proceedings of the 18th International Conference on Software Engineering, Berlin, Germany, 1996, pp: 442 – 449