Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

An Analysis of the History of Classical Software Development and Agile Development

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

Armin Eberlein
Department of Computer Science & Engineering
American University of Sharjah,
Sharjah, UAE
eberlein@ucalgary.ca

Abstract: **The ongoing debate over the merits of classical Software Engineering (SE) methodologies and agile methodologies has so far resulted in no clear benefits for the SE community. This paper uses the CHAPL[1] framework developed in our previous research to analyse the history of classical SE methodologies and agile methodologies. Our historical analysis focuses on the following three perspectives: the practices, the principles, and the technological context. The analysis reveals that both approaches to software development have similar roots and that their proponents have ample ground for constructive discussions. In fact, both approaches can be seen as complementary and their integration could contribute to project success.**

Keywords: software engineering, software process, traditional software engineering, agile development, historical analysis.

I. INTRODUCTION

Significant effort has been invested over the last decades in identifying good practices, models and methods[2] that lead to more efficient software development. The large number of methodologies has also led to heated debates amongst software developers who now tend to classify SE methodologies into two categories:

1) Classical SE methodologies: They are also often referred to as heavy-weight or plan-driven and require upfront requirements definition, documentation and detailed plans. Two prominent examples are the waterfall model [26] and the spiral model [33]. Larger frameworks, such as the CMM, generally support classical SE methodologies [1].
2) Agile methodologies: They are often called light-weight or agile. This category includes e.g., (Extreme Programming) XP [2] and Scrum [58], which follow the 12 agile principles as described in Beck [2] and Cockburn [3].

These two categories appear to have conflicting ideas and are supported by two groups of proponents that have rarely engaged in constructive discussions. With the continuous growth of size and complexity of software applications, we believe it is time to systematically examine the differences and relationships between the two categories and their associated development philosophies. Our initial research shows that there are multi-dimensional relationships between SE methodologies. The five dimensions that we use in our analysis are Contextual, Historical, Analysis by Analogy, Phenomenological, and Linguistic [4]. They are typical analysis methods used in philosophy [7; 8]. This paper focuses on the historical dimension.

We believe that a better understanding of the historical links between the different SE methodologies is of great value to the SE community. Klein states that historical analysis increases awareness of the shared history, and a shared history can reduce hostility, increase commitment and make communication easier across boundaries [9]. It also supports reflection and avoids dogmatism as witnessed by many of us [10]. Therefore, this paper makes an initial step towards addressing the historical link between the methodologies. To achieve this, we selected and analysed over 100 books and papers on SE methodologies published at major IEEE SE conferences and journals, such as ICSE conferences, IEEE Transactions on SE, and ACM Transactions on SE and Methodology. The research described in this paper focuses on the historical analysis of some typical practices, principles[3] and technological context of methodologies from the two categories discussed above.

The rest of the paper is organised as follows: The historical analysis of the practices used in classical SE and in agile development is presented in Section II, and the principles in Section III. Analysis of the technological context of SE methodologies is presented in Section IV. Related work is described in Section V. Conclusions and discussion of future work are summarised in Section VI.

II. HISTORICAL ANALYSIS OF THE *PRACTICES* USED IN CLASSICAL SE AND AGILE DEVELOPMENT

The terms “agile” and “agility” can be traced back to the manufacturing industry in 1991 when “lean development” emerged in manufacturing with the aim of eliminating waste, amplifying learning, delivering as fast as possible and empowering teams [11]. Youssef [12] even coined the term “agile manufacturing” around that time. It therefore appears that the roots of agile methodologies can be traced back to traditional engineering disciplines.

The idea of iterative and incremental development used in most agile process models can be found as early as the 1930s when a quality expert at Bell Labs used this practice to improve product quality [13]. A complete definition and explanation of

[1] See [4] for more information

[2] For a detailed discussion on the differences between the terms SE method, SE methodology, software process model and software lifecycle, please see [5, 6].

[3] According to the Webster English Dictionary, a principle is a comprehensive and fundamental law, doctrine, or assumption. In this paper, we define a principle as an idea, insight or objective behind a practice, while practices are the activities that carry out the principle. A good discussion about the differences between principle and practice can be found in [17].

this practice for software development is given by Basili in 1975 [14]. Gladden and Gilb have proposed the practice of “delivering working software early” in the early 1980s [15, 16] to address the issue of late delivery of software products resulting in customer dissatisfaction. These practices can be directly mapped to the agile practice “frequent delivery” and “continuous integration” (see Table 1).

The practice of using a prototype of working software as the primary measure of progress is one of the principles in agile modelling. A similar idea can be found in manufacturing as early as 1982 when Gladden stated that a physical object conveys more information than a written specification [15]. Prototyping has been extensively used in classical SE for feasibility studies and elicitation of requirements. For instance, Rapid Application Development (RAD) is a development approach that is based on this practice [18].

Requirements engineering (RE) practices are a very important part of most engineering disciplines since having a good understanding of requirements is the first step towards delivering a product that meets customer expectations. However, RE is not an easy process as uncertainty and volatility of requirements are two problems that challenge developers. Joint Application Design (JAD) is a practice used in IBM Research Labs in Toronto in the late 1970’s [19]. The fundamental objective of this practice is to get quality business requirements through active participation of stakeholders. Some ideas used in JAD are similar to stakeholder collaboration used in Dynamic Systems Development Method (DSDM) [20], another representative of agile methodologies.

The importance of having a work environment that helps improve communication among team members has been discussed by Weinberg in [21]. He argued that face-to-face communication can help exchange information efficiently. The practice in XP of having an “open workspace” also emphasizes the importance of face-to-face communication.

The historical analysis of practices used in agile development and traditional SE shows that there are clear similarities and significant overlap in the use of practices. Our findings based on the historical analysis of practices is summarised in Table 1.

III. Historical Analysis of the *Principles* used in Classical SE and Agile Development

The ultimate goal of SE theories, techniques and methodologies is to help developers produce high-quality products in an economical and timely manner [22]. However, there is not one ideal process that works in all situations. In fact, development processes and techniques tend to change at least slightly each time when a new product is designed and produced [23] even within the same company. Dijkstra states that any piece of software to be developed is new and its development process is an innovation process [24]. Thus, the software process has to be customized according to the specific needs of the project [25].

In this section, we want to examine the fundamental principles used in classical SE and in agile development irrespective of the terminology used. This is because those SE principles tend to be enduring [17] and can be instantiated into different practices.

In [26] where the classical waterfall model was presented, Royce emphasized that a designer “must communicate with

TABLE 1. LINKS BETWEEN PRACTICES USED IN TRADITIONAL SOFTWARE DEVELOPMENT AND AGILE DEVELOPMENT

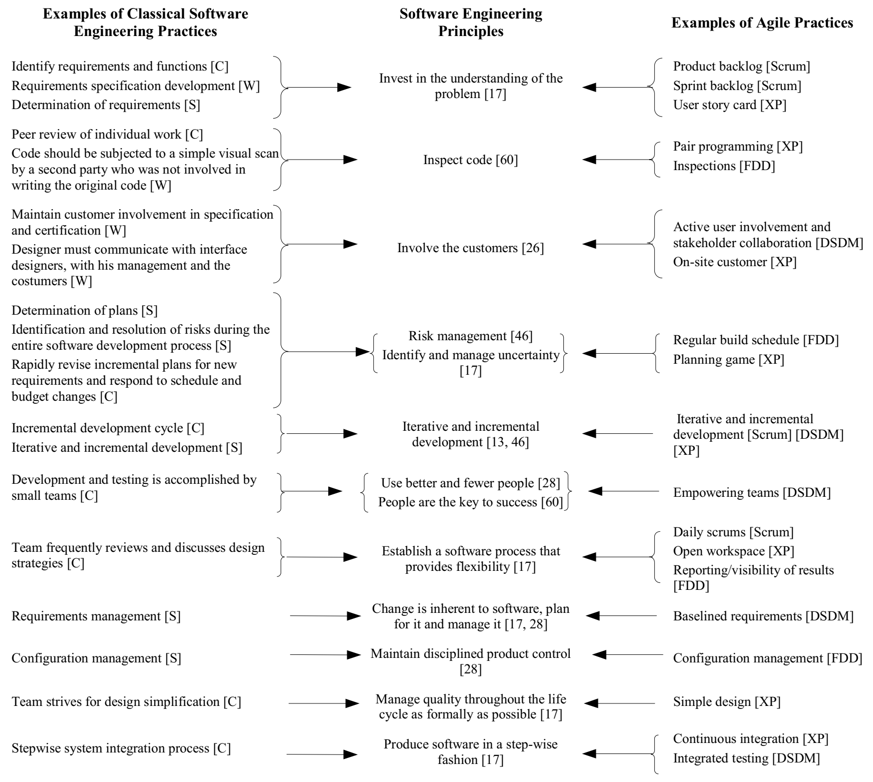
Practice in Traditional Software Development and Other Disciplines (shown in italics)	Practice in Agile Development
Delivering working software early [15, 16]	Continuous integration [XP]
	Frequent delivery [DSDM]
Iterative development [13] [14]	Iterations and increments [Scrum] [DSDM] [XP]
A "flexible" approach that embraces change is preferable [43] Accommodate rapid requirements change [44] [45]	Product backlog [Scrum]
Get the software into production in a matter of weeks [46]	Small releases [XP]
Iteratives development [13] [14]	Continuous integration [XP]
	Sprint backlog [Scrum]
	Iterations and increments [DSDM]
People who occupy a building should (in conjunction with a professional) be the ones to make the high-impact decisions [47]	Active user involvement (DSDM)
	On-site customer [XP]
The motivation of developer is the inner directing force for designing better software [21]	Empowered teams [DSDM]
1) An adequate environment contributes to the improvement of quality [21] 2) Face-to-face communication helps transmit information efficiently [21]	Open workspace [XP]
Evolution of a plan in response to business or technical changes [48]	Planning game [XP]
	Daily scrums [Scrum]
	Regular build schedule [FDD]
Software Configuration Management [49]	Configuration management [FDD]
	Reversible changes [DSDM]
Test-driven design iterations [43]	Tests-driven development [XP]
Integrated testing [50]	Integrated testing [DSDM]
Baselined requirements [51][52]	Baselined requirements [DSDM]
Involving stakeholder in design [19]	Stakeholder collaboration [DSDM]
Better software design [24]	Refactoring [XP]
Software inspections [53]	Inspections [FDD]
	Reversible changes [DSDM]
Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away [54]	Simple design [XP]
	Fitness (DSDM)
1) *Self-organizing teams increase the speed of new product development* [55] 2) *Self-managing groups can be expected to be more successful in turbulent environments* [56] 3) Self-organizing teams create a culture of innovation [57]	Self managed teams [Scrum]
	Empowering teams [DSDM]
	Feature teams [FDD]

Notes: (1) [XP], [Scrum], [DSDM], [FDD] represent eXtreme Programming [2], Scrum [58], Dynamic Systems Development Method [20], Feature Driven Development [59] respectively
(2) The references in the left column are the sources that the practices or principles of traditional Software Development have been discussed

interfacing designers, with his management and the customers” and needs to “maintain customer involvement in specification and certification”. This principle shares the same values with the practices of “Active user involvement” and “stakeholder collaboration” in DSDM [27], and “On-site customer” in XP. As another example, “using better and competent people in software development” is one of the prominent principles of agile, which has also been discussed as a principle in a lot of classical SE literature (e.g., [28; 29; 24]).

Furthermore, the underlying principle behind “Product backlog” and “Sprint backlog” in Scrum and “User story card” in XP is similar to requirements specification in waterfall, cleanroom [30] and spiral models. All of them assume that at least some requirements need to be known before implementation can start. However, there is significant disagreement in practice between the two schools of thought on the extent to which requirements have to be determined and documented. For a big project with high requirements availability and low requirements volatility it is useful to first get a clear picture of all system requirements. However, for a smaller or a medium sized project with uncertain requirements it makes sense to scale down requirements specification to its minimum format by using incremental and iterative processes that help discover requirements gradually.

Based on the analysis of some fundamental principles used in classical SE methodologies and agile methodologies, we can see


Examples of Classical Software Engineering Practices
Software Engineering Principles
Examples of Agile Practices
Identify requirements and functions [C]
Requirements specification development [W]
Determination of requirements [S]
Invest in the understanding of the problem [17]
Product backlog [Scrum]
Sprint backlog [Scrum]
User story card [XP]
Peer review of individual work [C]
Code should be subjected to a simple visual scan by a second party who was not involved in writing the original code [W]
Inspect code [60]
Pair programming [XP]
Inspections [FDD]
Maintain customer involvement in specification and certification [W]
Designer must communicate with interface designers, with his management and the costumers [W]
Involve the customers [26]
Active user involvement and stakeholder collaboration [DSDM]
On-site customer [XP]
Determination of plans [S]
Identification and resolution of risks during the entire software development process [S]
Rapidly revise incremental plans for new requirements and respond to schedule and budget changes [C]
Risk management [46]
Identify and manage uncertainty [17]
Regular build schedule [FDD]
Planning game [XP]
Incremental development cycle [C]
Iterative and incremental development [S]
Iterative and incremental development [13, 46]
Iterative and incremental development [Scrum] [DSDM] [XP]
Development and testing is accomplished by small teams [C]
Use better and fewer people [28]
People are the key to success [60]
Empowering teams [DSDM]
Team frequently reviews and discusses design strategies [C]
Establish a software process that provides flexibility [17]
Daily scrums [Scrum]
Open workspace [XP]
Reporting/visibility of results [FDD]
Requirements management [S]
Change is inherent to software, plan for it and manage it [17, 28]
Baselined requirements [DSDM]
Configuration management [S]
Maintain disciplined product control [28]
Configuration management [FDD]
Team strives for design simplification [C]
Manage quality throughout the life cycle as formally as possible [17]
Simple design [XP]
Stepwise system integration process [C]
Produce software in a step-wise fashion [17]
Continuous integration [XP]
Integrated testing [DSDM]

Legend: 1. [W], [S], [C] represent Waterfall model, Spiral model, and Cleanroom methodology respectively. Refer to Table 1 for the meanings of [DSDM], [XP], [FDD], and [Scrum] 2. The arrows show the link of practices of classical SE and agile development through SE principles.

Figure 1. Links between SE principles used in classical SE and agile development

that agile methodologies have at least some historical links to classical SE methodologies. This is a further indication that we should not dogmatically separate classical SE from agile development. Fig. 1 illustrates the results of our analysis of the historical links between the practices of the two schools through fundamental SE principles.

IV. Historical Analysis of the Technological Context of Classical SE and Agile Development

In this section, we want to investigate how technology influenced the development and use of SE methodologies over the years. Some researchers have argued that context is the key reason for deciding which methodology is suitable [10, 31]. One needs to consider historical, organisational, cultural and project contexts.

In this research, we analysed the history of the technological context of several classical SE and agile methodologies. For the purpose of this paper, the discussion of context focuses on the technology available when a SE methodology was developed, used, and evolved. We will take several methodologies as examples.

A. Technological Context of the Waterfall Model

The waterfall model emerged in the late 1960s when programming languages were inefficient and hardware consisted of mainframe computers with slow CPUs and very limited memory. The waterfall model mainly addresses management issues which include well-defined development phases and some basic SE principles, such as involving customers and producing quality requirements specifications. It is suitable within the technological context of that time and for large, long-term projects that require extensive upfront contracts between developers and customers. However, after contracts have been established, requirements are not supposed to change any more. Technical writers would be hired to handle the large amount of documentation required for the project. A librarian would keep track of code libraries and other paper documents as electronic storage space was expensive and no document management systems were available [29]. Having well-defined requirements documents was an important means for effective communication among developers in big teams [32]. With the availability of configuration management and documentation tools nowadays, the instantiation of the waterfall model in practice has to change.

B. Technological Context of Agile Methodologies

When agile methodologies emerged in software engineering, the technological landscape had drastically changed:

- Very powerful PCs are available at low cost
- Storage space is easily available
- Network facilities have increased capabilities
- Very powerful object-oriented languages, such as C++, Java and J2SE techniques, have been developed
- Internet and Web technologies are widely available
- Visual programming technologies and highly interactive GUI

and programming environments help design interactive user interfaces and allow the fast implementation of prototypes.

These advancements increased the ambitions of the software industry and enabled developers to deal with a wide range of applications from small software projects that need 2 to 3 developers for 3 to 6 months to extremely complex and large systems that require hundreds or even thousands of developers. Over the last decade, software developers have found that it is hard to plan the entire system development process in advance due to the fact that many systems, especially web-based or web-related systems, change and grow rapidly in their requirements, functionality and contents during their life cycle, much more than what they encountered before [36].

To deal with the challenge of rapid change, agile methodologies were developed. They could immediately address some challenges of many small to medium-sized software projects. More importantly, these methodologies evolved and matured into a new category of SE technologies with tremendous possibilities to maximise developers' potential [37]. They are now supported by new technologies that allow them to:

- model requirements and system behaviour early on
- accommodate requirements change at much lower cost and manage and control requirements changes with support of requirements management tools
- conduct automated testing with the support of tools
- easily contact and involve customers throughout the world using mobile phones, video conferencing, and high-speed internet
- manage and plan software projects very efficiently with the support of project management tools
- produce new software releases within a short period of time using efficient and effective programming and configuration management tools.

Furthermore, the advent of J2EE, .NET and various other technologies provides support to developers to apply agile practices in software projects.

The detailed analysis of the historical context of other SE methodologies shows that the technological context and the project context play important roles in the emergence and evolution of SE methodologies. A summary of our research results is given in Table 2.

V. Related Work

To the best of our knowledge, very little research has investigated in depth the historical links between classical SE methodologies and agile methodologies. The closest related work was done by Abbas *et. al.* [38] who studied the roots of some agile practices. Abbas *et. al.* argued that some agile practices have their historical roots in older SE practices. However, their work did not investigate links between classical and agile SE considering the principles and the technological context.

Other related work is the work by Larman and Basili [13] who discussed the historical roots of the practice "iterative and incremental methodology" in great detail. Abrahamsson *et. al.* conducted a comparative analysis of several agile methodologies and their relationships [39]. However, no historical analysis of the relationship between classical methodologies and agile methodologies is provided. Turner and Jain show that agile practices support 11 Process Areas and Generic Practices of CMMI which indicates some links between CMMI and agile development [40]. We hope this paper provides a more comprehensive analysis of this issue.

TABLE 2. HISTORICAL ANALYSIS OF THE TECHNOLOGICAL CONTEXT OF MAJOR SOFTWARE DEVELOPMENT METHODOLOGIES

Time period	Name of major new methodologies emerged	Technologies available		Examples of types of software projects
		Examples of several major program languages and/or SE tools used	Examples of computer hardware	
Before 1968	Waterfall [26]	(1) Languages: first, second and partial third generation languages such as FORTRAN, ALGOL, COBOL, BASIC, PL/I (2) Tools: No significant tool support available	● Mainframe computers ● Storage space is very limited, slow and expensive.	Large software projects (e.g. military projects), system software, new languages, and operation systems
1970-1979	● Structured programming [61] ● Modularization & information hiding [62] ● Abstract Data Types [63] ● Model data and algorithms separately [64]	(1) Languages: third generation language, e.g., C, FORTRAN, COBOL (2) Tools: ● Compilers run on microcomputer ● Unix system, DOS ● Some text editors for program languages ● Some tools provided isolated support for single activity, like editing programs, debugging, etc. [73]	● Microcomputers and workstations are available ● Storage space is still very limited ● Other hardware facilities are available such as various monitor, printers, interfaces, etc.	● Software projects increase size and complexity. ● Information systems increase size and complexity. ● Computer games ● Large number of software projects of medium size and complexity
1980-1989	● Structured Systems Analysis and Design Method (SSADM) [65] ● Prototyping [34, 35] ● JAD [19] ● Evolutionary development [16] ● Spiral model [33]	(1) Languages: third and fourth generation languages such as: Pascal, C, Ada, dBASE II, and Foxbase (2) Tools or Environments: ● Unix, MSDOS, Windows, ● Increased computational power of PC ● Graphical user interfaces (GUI) that have tremendous commercial impact at SE communities (starting in mid-1980s) ● Increasing number of compilers, program editors and debugging tools ● Some interactive programming environments	● Increased power and popularity of Personal Computers (PCs) and workstations ● Storage space is easily available, but still constraint ● The functionality and efficiency of hardware facilities were increased	Software industry grows very fast [74] ● The size and complexity of many software projects continue to grow. ● Information systems with further increasing size and complexity. ● Increasing popularity of PC applications ● Computer games with increasing size and complexity
1990-1999	● Capability Maturity Model (CMM) [66, 67] ● ISO 9000-3 [68] ● Object-Oriented SE[69] ● Rapid Application Development [18] ● Scrum [58] ● DSDM [20] ● Synch and stabilize process [70] ● XP [2, 71] ● Feature driven development [59]	● C++ ● PowerBuilder is an effective tool to construct interactive interfaces that help implement prototypes quickly ● Informix-4GL ● Components engineering and middleware ● Script languages ● Java and J2SE ● Internet and Web technologies emerged ● "Open source paradigm" started [77] ● Visual programming technologies and highly interactive GUI. ● Integrated Software Development Environments [73; 76]	● Powerful PCs are available at much lower cost and much higher capability ● Storage space is easily available ● Network facilities available ● The functionality and efficiency of hardware facilities continue to increase	Software industry continue to grow very quickly [74] ● The size and complexity of software projects continue to grow. e.g. space projects ● PCs applications keep becoming more popular and powerful ● Various business software applications (including MIS system) associated with internet applications flourish ● Many software projects are related to web applications
2000-Now	● CMMI ● Rational Unified Process (RUP) [72] ● Agile modelling [78]	● Proliferating of open source development [75] ● Web technologies were improved very quickly ● Middleware, components, and COTS technologies ● J2EE and Microsoft's .NET technology ● Rational Rose ● Intellectual visualized GUI and SE environments ● Powerful script languages and visual environments for web application development	● Very powerful PCs with enormous hardware and software resources ● Huge storage space is available at low cost ● Very powerful internet facilities ● The functionality and efficiency of hardware facilities continue to improve	● Software projects associated with military purposes ● Software products used almost everywhere ● Projects related to everyday life ● Large number of software projects that are related to advanced web application

Note: The time period of the methodologies and techniques emergence shown in the table refers to the time that they became popular.

VI Conclusion and Future Work

Unfortunately, all SE methodologies still have significant limitations [41]. Thus, we should stand back and look at the origin of these methodologies, the philosophy behind the methodologies, and the technologies and social environment which supported the generation of these methodologies. Research into the historical links between classical SE methodologies and agile methodologies can help us understand the relationships between the practices in the two development philosophies. This paper presented our findings of the historical links from three perspectives: the practices, the principles, and the technological context.

A number of implications of this investigation are summarised below:

- There is significant evidence that practices used in SE methodologies and in agile methodologies have historical links. Moreover, many practices in both approaches have roots in other disciplines as well as in traditional engineering disciplines as shown in Table 1.
- An analysis of the fundamental principles behind some practices used in agile methodologies shows that the same principles are used in the practices of classical SE methodologies, i.e. we can see links between the fundamental principles behind practices in both categories of methodologies.
- The technological context has a significant impact on the emergence, evolution and change of methodologies. A methodology that works well in the context of one organization with its particular social, technological, managerial and cultural environment does not guarantee that it will also work well in another context. This means that any SE methodology has to be adapted to the context of its usage.

Historical links between existing methodologies indicate that classical SE methodologies and agile methodologies share the same values, and therefore, are all valuable assets of the SE discipline. This holistic view is very important since it will:

- provide information about the relationships between SE methodologies which can provide developers a good position to understand deeply, judge objectively and use SE methodologies wisely in practice,
- allow developers to study valuable practices of all SE methodologies and integrate them to address today's SE problems. The benefits of integrating practices used in classical SE and in agile methodologies have already been reported by Boehm in [42].

We argue that the disparities between SE methodologies reflect the fact that different practices are required to tackle the different challenges in a large array of software projects. The failure of a software project is not caused by a software methodology but by the development team that selected an inappropriate development methodology.

Already in 1996, Basili emphasized that we need models that help us reason about the suitability of SE practices and methodologies for a specific software project [41]. However, still very little research has been done in this area [31]. We argue that, based on an understanding of the relationships between classical SE and agile methodologies, it is possible and beneficial to SE to develop a reasoning mechanism that assists with the selection of best practices, techniques and methodologies for software projects.

References

[1] M.C. Paulk, C.V. Weber, B. Curtis, M.B. Chrissis, "The Capability Maturity Model: Guidelines for improving the software process",. Addison-Wesley, Reading, Mass., 1995.

[2] K. Beck, C. Andres, "Extreme programming explained: Embrace change". Addison-Wesley, Boston, 2005.

[3] A. Cockburn, "Agile software development, the people factor", Computer, 34 (11), p.131-133.

[4] L. Jiang, A. Eberlein, "Towards a framework for understanding the relationships between classical software engineering and agile methodologies". In: APSO 2008,:In conjunction with 30th International Conference on SE, Leipzig, Germany, 10 – 18, May 2008.

[5] C. Floyd, "A comparative evaluation of system development methods". In Olle, T.W., Sol, H.S., and Verrijn-Stuart, A.A., (eds) (North-Holland, Amsterdam), pp. 19-54, 1986.

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

[7] B. Dahlbom, L. Mathiassen, "Systems development philosophy", ACM SIGCAS, Computers and Society, Vol. 22 , Issue 1-4, pp: 12 – 23, 1992

[8] C. Mitcham, "The importance of philosophy to engineering", Teorema XVII(3):pp. 27-47, 1998

[9] H. Klein, R. Hirschheim, "The structure of the IS discipline reconsidered: Implications and reflections from a community of practice perspective", Information and Organization, 18 (4), p. 280,. 2008

[10] P. Kruchten, "Voyage in the agile memeplex" ACM Queue: Tomorrow's Computing Today, 5 (5):38–44, July 2007.

[11] M. Poppendieck, T. Poppendieck, "Lean software development: An agile toolkit", Addison Wesley, The Agile Software Development Series, New York, NY., 2003.

[12] M.A. Youssef, "Agile manufacturing: a necessary condition for competing in global markets", Industrial Engineering, Dec., 18(20), 1992.

[13] C. Larman, V.R. Basili, "Iterative and incremental development: A brief history", IEEE Computer, 36(6): 47-56, 2003.

[14] V. Basili, J. Turner, "Iterative enhancement: A practical technique for software development" IEEE Trans. Software Eng., pp. 390-396, 1975.

[15] G.R. Gladden, "Stop the life-cycle: I want to get off", Software Engineering Notes, 7(2): pp. 35-39, 1982.

[16] T. Gilb, "Evolutionary delivery versus the waterfall model", ACM SIGSOFT Software Engineering Notes, 10(3): pp. 49-61, 1985.

[17] P. Bourque, "Fundamental principles of software engineering–A journey", Journal of Systems and Software, 62 (1), p. 59, 2002.

[18] J. Martin, "Rapid application development" Macmillan Publishing, New York, 1991.

[19] J. Wood, D. Silver, "Joint application development", John Wiley and Sons, NY., 1989.

[20] D. Millington, J. Stapleton, "Developing a RAD standard", IEEE Software, 12(5), 54-56, 1995.

[21] G.M. Weinberg, "The psychology of computer programming", New York : Van Nostrand Reinhold, 1971

[22] G. Cugola, C. Ghezzi,"Software processes: A retrospective and a path to the future", Journal of Software Process - Improvement and Practice, vol. 4, pp. 101-123, 1998.

[23] T. Jarratt, C. Eckert, P.J. Clarkson, "Pitfalls of engineering change, change practice during complex product design", Advances in Design, Springer London, 2006.

[24] E.W. Dijkstra, "The humble programmer", Communication of the ACM, 15(10): p. 859-866, 1972.

[25] R.L. Glass, "Matching methodology to problem domain". Communications of the ACM, 47 (5), p. 19, 2004.

[26] W. Royce, "Managing the development of large software systems". Proceedings of Westcon, IEEE CS Press, pp. 328-339, 1970.

[27] J. Stapleton, "Dynamic systems development method", Addison Wesley, Longman, Reading, Mass., 1997.

[28] B. Boehm, “Seven basic principles of software engineering”, State of The Art Report on SE Techniques, Infotech International Ltd., Mmdenhead, UK, 1976.

[29] M.V. Zelkowitz “Perspectives on software engineering”, Computing Surveys. (ACM) 10(2), 197-216, 1978.

[30] S.J. Prowell, C.J. Trammell, R.C. Linger, J.H. Poore, “Cleanroom software engineering: technology and process”, Addison-Wesley, 1999.

[31] L. Jiang, A. Eberlein, B.H. Far “A case study validation of a knowledge-based approach for the selection of requirements engineering techniques”, Journal of Requirements Engineering, 13(2), pp. 117-146, 2007.

[32] F.P. Brooks, “The mythical man-month: Essays on software engineering”, Addison-Wesley, 1995.

[33] B. Boehm, “A spiral model of software development and enhancement”, Computer, May, pp. 61-72, 1988.

[34] H. Gomaa, D.B.H. Scott “Prototyping as a tool in the specification of user requirements”, International Conference on Software Engineering, p. 333, 1981.

[35] H. Gomaa, “The impact of rapid prototyping on specifying user requirements", Software Engineering Notes, 8 (2), p. 17, 1983.

[36] A. Ginige, S. Murugesan, “Web wngineering: an introduction”, IEEE Multimedia, Special Issues on Web Engineering, vol 8, no 1, pp 14-18, 2001.

[37] R. Cowan, “Software wngineering technology watches”, IEEE Software, 19 (4), p. 123, 2002.

[38] N. Abbas, A. Gravell, G. Wills “Historical roots of agile methods: where did “agile thinking” come from?” In Proceedings of Agile Processes and eXtreme Pogramming in Software Engineering, Limerick, Ireland, 2008.

[39] P. Abrahamsson, J. Warstab, M.T. Siponenb, J. Ronkainena, “New directions on agile methods: A comparative analysis”, Proceedings of the 25th International Conference on Software Engineering, pp. 244–254, 2003).

[40] R. Turner, A. Jain, “Agile mMeets CMMI: culture clash or common cause?” In D. Wells and L. Williams (Eds.): LNCS , XP/Agile Universe 2002, pp. 153–165, 2002.

[41] V.R. Basili, “The role of experimentation in software engineering: past, current, and future”, 18th International Conference on SE, Berlin, Germany, pp: 442 – 449, 1996.

[42] B. Boehm, “Using risk to balance agile and plan-driven methods”, IEEE Computer, 36 (6), p. 57, 2003.

[43] S. Thomke, “The role of flexibility in the development of new products: An empirical study”, Research Policy, 26, pp. 105–119, 1997

[44] M. Aoyama, “Agile software process and its experience”, 20th international conference on software engineering, 1998.

[45] M. Aoyama, “Web-based agile software development”, IEEE Software, 15 (6), p. 56, 1998.

[46] T. Gilb, “Principles of software engineering management”, Addison-Wesley, 1988.

[47] C. Alexander, “The timeless way of building”, Oxford University Press, New York, 1979.

[48] H. Takeuchi, I. Nonaka, “The new product development game”, Harvard Business Rev., Jan./Feb., pp. 137-146, 1986

[49] E. Bersoff, V. Henderson, S. Siegel, “Software configuration management”, Prentice-Hall, Englewood Falls, NJ, 1980.

[50] I. Sommerville “Software engineering”, Addition Wesley, 8/e, 2006.

[51] T.E. Bell, T.A. Thayer “Software requirements: are they really a problem?” 2nd International Software Engineering Conference, October 1976.

[52] M.C. Paulk, B. Curtis, M.B. Chrissis, “Capability maturity model for software”, Software Engineering Institute, CMU/SEI-91-TR-24, ADA240603, August 1991.

[53] M.E. Fagan, “Design and code inspections to reduce errors in program development”. IBM System Journal, 15(3), pp. 182–211. 1976.

[54] A.D. Saint-Exupery, “Wind, sand and stars”, London: Heinemann, 1954.

[55] D.D. Dill, A.W. Pearson, “The self-designing organization: structure, learning, and the management of technical professionals” Technology Management: the New International Language (p. 33), 1991.

[56] C. Smith, D. Comer “Self-organization in small groups: A study of group effectiveness within non-equilibrium conditions”, Journal of Human Relations. 47:553–81, 1994.

[57] K. Imai, I. Nonaka, H. Takeuchi, “Managing the new product development Process: How Japanese companies learn and unlearn”, In K.B. Clark, R. H. Hayes. and C. Lorenz (eds.), The Uneasy Alliance: 337-376, Boston: Harvard Business School Press, 1985.

[58] K. Schwaber, “Scrum development process”, In 10th Annual ACM Conference on OOPSLA 1995, Austin, Texas, USA, 117-134, 1995.

[59] S.R. Palmer, J.M. Felsing, “A practical guide to feature driven development”, Upper Saddle River, NJ: Prentice Hall, 2002.

[60] A.M. Davis, “Fifteen principles of software engineering”, IEEE Software, Volume: 11, Issue: 6, pp: 94-96, 1994.

[61] O.J. Dahl, E.W. Dijkstra, “C.A.R. Hoare, “Structured programming”, Academic Press, London; New York, 1972.

[62] D.L. Parnas, “Abstract types defined as classes of variables”, ACM SOGMOD Record, Vol. 8, No. 2, pps. 149-154, 1976.

[63] B. Liskov, S. Zilles, “Programming with abstract data types”, Proc. ACM SIGPLAN Symposium, pps. 50-59, 1974.

[64] M.A. Jackson, “Principles of program design”, London; NY : Academic Press, 1975.

[65] J.R. Cameron, “JSP & JSD: the Jackson approach to software development”, IEEE Computer Society Press, 1983.

[66] W.S. Humphrey, “Managing the software process”, Addison-Wesley Longman Publishing Co., Inc. 1989.

[67] M.C. Paulk, B. Curtis, M.B. Chrissis, C.V. Weber, “Capability maturity model, version 1.1”, IEEE software 10 (4), p. 18, 1993.

[68] D. Hoyle, “ISO 9000 Quality Systems Handbook”, 2nd ed., Oxford: Butterworth-Heinemann, 1994.

[69] I. Jacobson, “Object-oriented software engineering: A use case driven approach”, Addison-Wesley, New York, 1992.

[70] M.A. Cusumano, R.W. Selby, “Microsoft secrets: How the world’s most powerful software company creates technology, shapes markets, and manages people”, The Free Press, 1995.

[71] K. Beck, M. Fowler “Planning extreme programming”, Boston: Addison-Wesley, 2001.

[72] P. Kruchten, “The Rational unified process”, 2nd ed., Addison Wesley, 2001.

[73] W. Harrison, H. Ossher, P. Tarr, “Software engineering tools and environments”, International Conference on Software Engineering, pp. 263–277, 2000.

[74] W.E. Steinmueller, “The U.S. software industry: An analysis and interpretative history”, in David C. Mowery (ed.), "The International Computer Software Industry," Oxford University Press, 1995.

[75] A. Mockus, R.T. Fielding, J.D. Herbsleb “Two case studies of open source software development: Apache and Mozilla”. ACM Transactions on Software Engineering and Methodology, 11 (3), p. 309, 2002.

[76] B. Myers, “Past, present, and future of user interface software tools”,. ACM Transactions on Computer-Human Interaction, 7 (1), p. 3. 2000.

[77] E. Raymond, “The cathedral and the bazaar. knowledge, technology, & policy”, 12 (3), p. 23, 1999.

[78] S. Ambler “Agile modeling: effective practices for extreme programming and the unified process”, John Wiley & Sons, Inc., New York, NY, 2002.