A Quantitative Framework for the Evaluation of Agile Methodologies

Karla Mendes Calo[1], Elsa Estevez[1,2], Pablo Fillottrani[1,3]
[1]Laboratorio de I&D en Ingeniería de Software y Sistemas de Información (LISSI)
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur,
Avenida Alem 1253,
(8000) Bahía Blanca, Argentina
2Center for Electronic Governance at United Nations University
International Institute for Software Development (UNU-IIST)
P.O. Box 3058, Macao SAR, China
3Comisión de Investigaciones Científicas de la Provincia de Buenos Aires
{kmca, ece, prf}@cs.uns.edu.ar

ABSTRACT

The methodologies for agile software development are fundamentally based on the collaboration with software users during the entire development process, the simplicity to adapt the product to changes in requirements, and on the incremental product delivery. Based on the Agile Manifesto, they have been accepted and are successfully used in projects where the detailed requirements are unknown at first and are identified during the development process from the interactions with the users and the feedback thus obtained. In this paper, we propose an evaluation framework for the methodologies for agile software development. This framework is applied in detail to two of them - Scrum and eXtreme Programming (XP). The definition of this quantitative framework is innovative, especially because it allows the evaluation of how the agile methodologies satisfy the basic principles defined by the Agile Manifesto, thus it can be used when deciding which methodology to adopt in a particular project.

Keywords: Agile Manifesto, Agile Methodologies, SCRUM, XP

INTRODUCTION

Traditionally, the software development processes put a marked stress on the process control. They define activities, devices and information to be produced, tools and notations to be used, orders to execute the activities, among other definitions. Although there are several development processes - Unified Process [1], V Process [2], etc, most of these processes derive from the Waterfall Model proposed by Boehm [3]. These processes, called traditional, have proven effective in large scale projects, particularly in regards to the administration of resources that can be used and the planning of the development time. However, the proposed approach by these methods is not the most adequate for the development of projects where system requirements change frequently, development times have to be drastically reduced and, at the same time, produce high quality products.

The Agile Methodologies appear as an alternative to the traditional methods of development. Keeping essential practices of the traditional methodologies, the agile methodologies focus on other dimensions of the project; for example: the collaboration with users during all stages of the development process and the incremental development of the software with very short iterations that provide a custom-made solution. The agile practices are specially indicated for products whose detailed definition is very hard to obtain from the beginning, or if defined, it would have a lesser value than if the product is built with a constant feedback during the development process.

The objective of this paper is to present an evaluation framework of agile methodologies that allow the evaluation of how the methodologies reach the values declared by the Agile Manifesto. The evaluation framework gives the opportunity to make a more informed decision when the time comes to select one of the methodologies. As an example, the framework is applied to the SCRUM and XP methodologies.

The rest of this work is structured as follows: Section 2 presents the Agile Manifesto and some of the commonly used agile methodologies. Section 3 explains two methodologies in detail: SCRUM and XP. After that, Section 4 presents and explains the evaluation framework, while Section 5 shows its application to SCRUM and XP. Finally, Section 6 presents a comparison with related works, conclusions and future works.

AGILE MANIFESTO AND AGILE DEVELOPMENT METHODOLOGIES

In February 2001, academics and experts of the software industry gathered in Utah, United States, in order to discuss values and principles that would facilitate a quicker software development and answers to the changes that might arise during the project. The idea was to offer an alternative to the processes of traditional development. As a result of this meeting, the Agile Alliance [4] was formed. This is a non-profit organization dedicated to promoting the concepts related to the agile development of software and helping organizations to adopt said concepts. The result of this meeting was a document known as the Agile Manifesto [5]. The Agile Manifesto includes four postulates and a series of associated principles. The postulates are:

1) *Value the individual and the development team's interactions above the process and the tools*. Three premises sustain this principle: a) team members are the main factor of a project's success; b) it's more important to set up a team than an environment. c) it's better to put a team together and to let it configure the environment based on its own needs.

2) *Value the software development that works over an exhaustive documentation*. The principle is based on the premise that documents can neither replace nor offer the added value that is achieved with direct communication between people through the interaction with prototypes. The use of documentation that generates works and does not add a direct value to the product must be reduced to the essential minimum.

3) Value the collaboration with the customer over the contractual negotiation. In agile development, the

customer is integrated and collaborates with the work team, just like any other member. The contract itself does not add value to the product; it is just a formalism that establishes lines of responsibility among parties.
4) *Value the answer to change over the follow up of a plan*. The speedy and constant evolution must be inherent factors to the development process. The ability to react to change over the ability to monitor and assure pre-established plans.
The development cycle applied by Agile Methodologies is iterative and incremental. This model allows the software to be delivered in small and usable parts, known as increments. Each iteration can be considered as a small project where activities such as requirement, analysis, design, implementation and testing are carried out with the objective of producing a subset of the final system. The process is repeated several times producing a new increment in ever cycle until the complete product is finished. Although all the agile methodologies adopt this cycle, each one of them presents its own characteristics.
The most commonly used agile methodologies are described as follows:
Scrum [6] – It is suitable for projects with a high ratio of change in requirements. Its main characteristic is the definition of sprints – each one of the repetitions of the process with a maximum duration of 30 days. The result of each sprint is an executable increment that is shown to the customer. Another relevant characteristic are the daily meetings that take place during the project. Said meetings do not require more than fifteen minutes from the development team and its objectives are the coordination and integration of the product to be delivered.
Crystal Methodologies [7] – They are a group of methodologies for software development characterized by the value of the people that compose the work team and the maximum reduction of the number of artefacts produced. It emphasizes on the efforts to improve the team members' skills and to define teamwork policies. The policies will depend on the size of the team, where a classification of colours will be established; for example, Crystal Clear corresponds to teams with 3-8 members and Crystal Orange to teams with 25-50 members.
Dynamic Systems Development Method (DSDM) [8] – It fulfils the general characteristics of defining an incremental and iterative process. It proposes five development stages: Viability Study, Business Study, Functional Modelling, Design and Construction, and Implementation. The iteration is produced during the last three stages. However, it foresees feedback in all of them.
Adaptive Software Development (ASD) [9] – It is a repetitive process, tolerant to changes and aimed at the software components. It defines three stages for the lifecycle: a) Speculation - the project starts and software features are planned; b) Collaboration - the product is developed; and c) Learning - the quality of the product is controlled and then it is delivered to the customer. The aim of the revision is to learn from mistakes made and to start the development cycle again.
Feature-Driven Development (FDD) [10] – It defines an iterative process with short iterations of two weeks maximum. The lifecycle consists of five steps: a) Development of a global model; b) Construction of a list of features (functions); c) Feature Planning; d) Feature Design; and e) Feature Construction.
Extreme Programming (XP) [11] – It defines an incremental and iterative process with continuous unit tests and frequent deliveries. The customer or a customer's representative is integrated to the development team. It recommends that the development of product functions is carried out by two people in the same post - pair programming. Before adding a new function, all found bugs must be corrected. Regression tests are constantly carried out in order to detect possible mistakes.

SCRUM AND XP – PRINCIPLES, ACTIVITIES, ROLES AND PRACTICES

The following two sections present the principles, activities, roles to be covered in the work teams and recommended Scrum and XP practices in detail.

Scrum

The methodology respects the evolutionary lifecycle and the iterative incremental delivery. At the beginning of the project, the functional and non functional requirements are identified and a list of such requirements called product backlog is made. The product backlog constitutes the base artefact to measure the project's progress. The iterations, called sprints deliver parts of the product called builds. Although they do not include all system functions, they constitute operational executables. Every iteration starts with an adapted planning guided by the customer and it ends with a demonstration of the customer's build. Every sprint can last a maximum of 30 days. In every sprint, the development team selects a group of higher priority items from the product backlog that turns into the development objective. The methodology proposes three stages:
1) *Planning Phase* – it is subdivided in: a) Planning - the development system, tools and the project team is defined and the product backlog is created with the list of requirements known at that time; priorities for the requirements are defined and the effort to carry out the implementation of those requirements is estimated; and b) the product architecture that allows the implementation of the specified requirements is defined.
2) *Development Phase* – it is the agile part, where the system is developed in sprints. Every sprint includes the traditional software development phases – requirements, analysis, design, implementation and delivery.
3) *Closure Phase* – it includes integration, testing and documentation. It indicates the implementation of all requirements, leaving the product backlog empty and the system ready to enter into production phase.
The methodology proposes the creation of self-managed and self-organized work teams, suggesting small teams that maximize the communication between its members. Within the work team, some roles are indentified, like the Scrum Master - responsible for assuring that the project is carried out based on Scrum rules, values and practices; the Product Owner - responsible for the project, administers, controls, maintains and publishes the product backlog; the Team Members - they have the authority to decide on the actions to take place and organize them in a way that allows the objectives of all sprints to be reached; and the Customer - it participates in the requirement-related tasks of the product to be developed, it provides ideas, suggestions and new needs.
Scrum foresees the following practices:
1) *Sprint Planning Meeting* – organized by the Scrum Master, it is divided in two stages. In the first stage, the customers, the owner of the product and the team members meet to decide about the objectives and functions of the new sprint. The second stage of the meeting takes place between the Scrum Master and the work team and it focuses on how the growth of the

product will be implemented during the process.
2) *Sprint* – it is a list of selected requirements to be implemented in the next repetition. The requirements are selected by the work team, together with the Scrum Master and the owner of the product during the meeting of the sprint planning. When all sprint items are completed, new system iteration is delivered.
3) *Scrum Daily Meetings* – they are run by the Scrum Master. They are basically organized in order to maintain a constant revision of the project progress. The members answer three questions: 1) What has been completed since the last meeting; 2) What obstacles or problems have been detected; and 3) What functions of the backlog are planned to be completed for the next meeting.
4) *Scrum Review Meeting* – the work team and the Scrum Master present the results of the sprint to the customer.
5) *Scrum Retrospective Meeting* – it takes place after finishing a product backlog and the revision of the sprint. The work team checks the fulfilment of the marked objectives at the start of the sprint. The necessary changes and adjustments will be analyzed and applied when necessary, the positive aspects will be stressed and the negative aspects will be changed, if possible in order to avoid repeating them in the next sprint.

eXtreme Programming

XP, formulated by Kent Beck, differs from the rest of the methodologies due to its stress on adaptability. The methodology is designed to offer the software that the user needs and when he needs it. The success of the methodology is based on boosting interpersonal relationships, promoting teamwork, continuous learning of the developers and a friendly working environment. The five basic principles of XP include:
1) *Simplicity* - simplify the design to speed up the development and to facilitate the maintenance through the updating of the code; 2) *Communication* - it encourages communication: written - like a self-documented code and joint tests, recommending the documentation of the class objectives and the functionality provided by methods; and oral - among programmers and with customers, recommending that both communication between both parties should be constant and fluent; 3) *Feedback* - it promotes the customer's constant feedback through short delivery cycles and demonstrations of the delivered functions; 4) *Courage* - to maintain simplicity by allowing the deference of design decisions; to communicate with others, even when this enables to show the lack of one's own knowledge, and to receive feedback during the development; and 5) *Respect* – should be instilled among team members - the developers cannot make changes that may cause the existing tests to fail or delay the work of fellow team members, and towards the work - the team members' main objective is to achieve a high quality product with an ideal design.
The development process consists on three stages:
1) *Interaction with the customer* – the customer permanently interacts with the work team. The initial requirement recollection phase is thus eliminated, and requirements are incorporated in an orderly fashion throughout the development. The methodology proposes using the User Story technique through which the user specifies function and non-function requirements of the product. Each history must be sufficiently atomic and understandable in order for the developers to implement the requirements in one iteration.
2) *Project Planning* – the work team estimates the required effort for implementing the user story. Each story must be implemented in a period of three weeks. Those stories that require more time are subdivided in order to be atomic and that they can be developed within the deadline.
3) *Design and Development of Tests* – the implementation is conducted by unit tests. Every time a function is going to be implemented, first the test must be defined and then the code to satisfy it. Once the code successfully completes the test, it is augmented and thereafter it continues. As the user stories are implemented, the small code fragments are integrated. In this way, a constant integration takes place, avoiding a more costly integration at the end of the project. XP promotes the programming in pairs, where the development is carried out by a pair of programmers. The pairs have to change periodically so that the knowledge can be acquired by the entire development group.
The defined roles for the team members include Programmer - in charge of writing single tests and producing the code; Customer - writes user stories and functional tests, assigns priorities to user stories and decides which ones will be implemented in each iteration; Tester - is responsible for tests, helps the customer to write functional tests, executes them, informs results to the rest of the team and maintains the support tool used to carry out tests; Tracker - provides feedback, verifies the degree of correctness of the project estimations and controls the project progress; Coach - is responsible for the whole process, guides the team for respecting XP practices and for executing the process correctly; Consultant - an external member of the team with specific knowledge of some subject necessary to solve problems that may arise during the project; and the Solicitor (big boss) - the link between the customer and the developers. He helps the team work effectively. His main task is coordination.
Among others, XP defines the following practices:
1) *Planning Game* – the team estimates the required efforts for implementing user stories.
2) *Updating* – ongoing activity for restructuring code. Its main objective is to remove code duplication, improve legibility and increase flexibility to facilitate changes.
3) *Pair Programming* – the development is carried out by a pair of developers.
4) *Constant Integration* – the code is integrated once it is available.
5) *In-situ Customer* – the customer must be present and available at all times.

EVALUATION FRAMEWORK

The proposed evaluation framework measures how agile methodologies fulfill the Agile Manifesto postulates described in Section 2. For this purpose, the framework defines measures that satisfy the measurement representational theory [12]. The measures are defined by using an interval scale [13].
The framework provides measurements for the four postulates presented in Section 2. These postulates (Pi, i=1..4) were expressed as the assessment of two attributes (Pi.1 y Pi.2). The measure of each postulate is defined as the sum of the measures of the related attributes, formulated as follows:

$$m(Pi) = m(Pi.1) + m(Pi.2) \quad i=1..4$$

For example, Postulate 1 (P1) - Value the individual and interactions of the development team over the process and the tools, it's measured adding the measure of how the methodology values the individual and the team interactions (P1.1) and the measure of how it values the process and the tools (P1.2).

The attribute that the principles try to stress (positive attribute) is measured in a scale of 0 to 5 and the other attribute (negative attribute) in a scale of -5 to 0. Therefore, each principle might obtain a measure of -5 – in case both attributes take the worst value (-5, the negative attribute and 0, the positive attribute), and 5 – in case both attributes take the best value (0, the negative attribute and 5, the positive attribute). If the result is a value of 0 or close to 0, it means that the methodology does not significantly value the positive attribute over the negative, which means that the Agile Manifesto postulate is not completely satisfied. The framework, the attributes and its measures are presented in Table 1.

Table 1. Evaluation Framework for Agile Methodologies

P1	*Value the individual and the team interactions over the process and the tools.*		
P1.1	*Value the individual and the interactions*	P1.2	*Value the process and the tools*
value	description	value	description
0	It does not define roles for individuals.	-5	It defines activities, deliverables, development and management tools.
1	Clear definition of roles for individuals.	-3	It defines activities, deliverables and development tools.
2	Clear definition of roles and responsibilities	-2	It defines activities and deliverables
3	Clear definition of roles, responsibilities and technical knowledge.	-1	It defines activities for each iteration.
5	Clear definition of roles, responsibilities, technical knowledge and interactions between members of the work team.	0	It defines project activities but not at the iteration level.
P2	*Value the software development that works over an exhaustive documentation.*		
P2.1	*Value the software development that works*	P2.2	*Value an exhaustive doncumentation*
value	description	value	description
0	Generate a deliverable at the end of the project.	-5	It requires detailed documentation at the beginning of the project.
3	Generate a deliverable with satisfactory testing at the end of each iteration.	-2	It only requires necessary documentation at the beginning of each iteration.
5	Generate a deliverable with satisfactory testing and integrated with the rest of the functions at the end of the iteration.	0	It does not require documentation to start implementing the functionality defined for an iteration.
P3	*Value the collaboration with the customer over the contractual negotiation*		
P3.1	*Value the collaboration with the customer*	P3.2	*Value the contractual negotiation*
value	description	value	description
0	The customer collaborates at the team's request.	-5	There exists a detailed contract and no changes are accepted.
3	The customer is part of the team. He answers to questions and plans the iterations.	-2	The contract demands considering changes during the project.
5	The customer is a team member, answers questions, plans iterations and collaborates in writing requirements and tests.	0	The contract does not add any value for the construction of the project products.
P4	*Value the answer to change over the monitoring of a plan*		
P4.1	*Value the answer to change*	P4.2	*Value the monitoring of a plan*
value	description	value	description
0	N changes are allowed during project execution.	-5	It defines a detailed plan at the beginning of the project.
1	Only high priority changes can be introduced during project execution.	-3	It defines a detailed plan of iterations and it does not accept changes during an iteration.
4	Evolution and change is recommended to be considered during iterations.	-2	It defines a detailed plan for each iteration, which can be modified.
5	Changes can be introduced during project iterations.	0	It defines no planning whatsoever.

FRAMEWORK APPLICATION

The application of the framework is shown in Table 2 and explained below.

Postulate P1. Scrum and XP obtain 5 in attribute P1.1 since both methodologies value the individual, define roles and responsibilities, and recognize the importance and promote the training of team members. Scrum obtains -3 in attribute P1.2 because it defines activities, deliverables and development tools; while XP obtains -2 because it only defines activities and deliverables. Conclusion: Scrum obtains 2 points and XP 3. Scrum satisfies P1 worst than XP since it defines development tools.

Postulate P2. Scrum obtains 3 points and XP 5 in the P1.2 attribute. The difference is that XP also considers partial integration of the software at the end of every iteration. Both methodologies are evaluated with a value of -2 for attribute P2.2 since both only require documentation for the planned iteration. Scrum and XP obtain a positive value for the P2 principle, with XP surpassing Scrum by 1 point. Conclusion: XP satisfies P2 better than Scrum because it requires the delivered increments to be constantly integrated with the rest of the functions.

Postulate P3. Both methodologies obtain the highest value in both attributes – 5 points for P3.1 and 0 for P3.2. Both consider the customer as a member of the team, someone who collaborates from the iteration planning, to the writing of requirements and functional tests. None of them use the contractual relationship to add value to the product. Conclusion: Both satisfy P3 in an optimum way.

Postulate P4. In attribute P4.1, Scrum obtains a value of 4 because even though it allows changes, they are not recommended during the current sprint. If a change in the current sprint is a priority, the required effort needs to be estimated again and, if necessary, remove tasks from the planned sprint. XP obtains the maximum value since changes can be incorporated during iterations. Due to a similar focus taking place with the planning, Scrum obtains a value of -3 and XP -2. Conclusion: XP obtains 3 points and Scrum 1. XP satisfies P4 better than Scrum.

Table 2. Applying the Framework to Scrum and XP

Postulates	P1			P2			P3			P4		
Methodology	P1.1	P1.2	total	P2.1	P2.2	total	P3.1	P3.2	total	P4.1	P4.2	total
Scrum	5	-3	2	4	-2	2	5	0	3	4	-3	1
XP	5	-2	3	5	-2	3	5	0	5	5	-2	3

RELATED WORK

In literature there are several works that compare agile methodologies. In regards to our knowledge, they are all based on qualitative comparisons. Abrahamsson et al [14] defines a list of key works and assesses several methodologies based on such list. The key words include: development state of the method, important points, special characteristics, adoption and the grade of support of the methodology for traditional activities of the development process. Iacovelli and Souveyet [15] define an assessment framework based on four high level attributes: capability to agility, use, applicability, and process and products. Strode [16] defines a comparison framework that includes the following attributes: methodology philosophy, models, techniques, tools, deliverables, practice and the degree of adaptability to a situation. Visconti and Cook [17] analyze how XP and Scrum satisfy the principles of the agile manifesto. After concluding that none of them completely satisfy the principles, they propose a methodology combining aspects of both. Despite almost all these frameworks somehow include an analysis about the way in which the methodologies fulfil the Agile Manifesto, all of them follow a qualitative approach.

Other studies take agile methodologies as references, and according to different approaches, provide frameworks that assess or measure different relevant aspects of the agile methodologies; for instance, the study using the Framework for Agile Method Classification [15] as reference. The approach used in this investigation intends to build a framework to classify agile methodologies through four views:

- *Usage* - why to use an agile methodology;
- *Capability to Agility* - What part of the agility is included in the method;
- *Applicability* - when the environment is favourable to use agile methodologies; and
- *Products and Process* - how the agility is expressed.

The views represent an aspect of an agile methodology that supports the selection of the method. Every method has been represented in the framework, taking into consideration the four previously presented views, plus a set of attributes for each view. This framework was applied to the most known methods, and the justification of their evaluation is completely documented in [21]. Its approach was based on which are the benefits of the presented aspects and what a favourable context would be like for its application in each compared methodology. Regarding the framework evaluation and the comparison of the methodologies, methods of similar characteristics were identified, based on the common attributes in some agile methodologies. Of these common characteristics derived from the framework, the agile methodologies were classified in three big classes: Software Development Practices Oriented Methods (Agile Modelling, Extreme Programming), Project Management Oriented Methods (Adaptive Software Development, Cristal Methodologies, Dynamic System Development Method, Scrum) and Hybrid Methods (Feature Driven Development).

Another proposal presented by Tsun Chow y Dac-Buu Cao is a survey study of critical success factors in agile software projects [18]. Its objective is to identify and provide information about critical success factors that will help software development projects to successfully use agile methodologies. It proposes a preliminary list of twelve possible identified critical success factors for each one of the four categories of the project's success - Quality, Scope, Time and Cost. This study was carried out throughout 109 agile software projects in 25 countries across the world, with organizations that also varied in size. These companies provided empirical information for an analysis that will lead to relevant conclusions. The contribution of this study is the reduction of the amount of anecdotic factors of success. According to this study, the only factors that could be called critical success factors are: (a) a correct delivery strategy, (b) an appropriate practice of agile software engineering techniques, (c) a high-calibrated team, (d) a good management of the agile development process, and (e) the active participation of the client in the project.

CONCLUSIONS

The main contribution of this paper is the definition of a quantitative evaluation framework to assess in which way agile methodologies satisfy Agile Manifesto postulates. Agile methodologies have their own characteristics and each emphasizes on specific aspects. Both selected methodologies promote such matters as teamwork, favouring interpersonal relationships among its members, boosting the fluent relationship with the client and generating the minimum documentation that contribute value to the project. Out of these resulting values, as we can see in Figure 2, we conclude that XP satisfies agile postulates better than Scrum. This framework allows us to quantify the adherence that both selected agile methodologies have for their comparison with each Agile Manifesto postulate.

This proposal differs from Framework for Agile Method Classification [15] because this one focuses on evaluating certain attributes, finding those that are common in both referenced agile methodologies, such as the size of the iterations, the size of the teams, and interactions with final users among other things, and from them, regrouping or classifying the methodologies in relevant classes to provide a support to select the best method, according to the context of the project. On the other hand, this proposed framework measures how the agile methodologies satisfy the Agile Manifesto postulates, no matter the context. If we complement both works, we could select the methodologies most suitable for a project based on the its environment, and from this set, the methodologies with higher adherence to the Agile Manifesto postulates.

Regarding the work A Survey Study of Critical Success Factors in Agile Software Projects [18], it concludes that the revealed critical factors in the study, obtained through empirical information, determine that independently from the used agile methodology, the list of attributes denominated as critical comprise part of the values declared by the Agile Manifesto and its postulates, and all lead to project success. The main difference with our framework is that this one groups or classifies critical success factors in six dimensions: correct delivery strategy, proper practice of agile software engineering techniques, team capacity, project processes, style of team work, and the client's participation as another team member - all of them in terms of Quality, Scope, Time and Cost. These critical success factors do not evaluate their impact on different agile methodologies.

Our future work includes extending the framework to measure the fulfilment of the Agile Manifesto principles, applying the framework to other agile methodologies and defining attributes to facilitate the choice of the most suitable methodology. In addition, we plan to extend this work by proposing the use of agile components on traditional methodologies. Thus, allowing the adaptation of favourable points of agile methodologies to traditional methodologies, making the latter more appropriate, flexible and scalable in projects where there might be certain risks due to change of requirements or impacts on predetermined business rules.

REFERENCES

[1] Jacobson, I., et al, The Unified Software Development Process, *Addison-Wesley* (1999).

[2] IABG, The V-Model, http://www.v-modell.iabg.de/.

[3] Boehm, B., Software Engineering, *IEEE Transactions on Computers*, 1226-1241 (1976).

[4] Agile Alliance, http://www.agilealliance.org/.

[5] Beck, K., et.al, Manifesto for Agile Software Development, http://agilemanifesto.org/.

[6] Scrum Alliance, http://www.scrumaliance.org

[7] Cockburn, A, Crystal Clear a Human-powered Methodology for Small Teams, (2004).

[8] Stapleton J. "DSDM Dynamic Systems Development Method: The Method in Practice". *Addison-Wesley*, (1997).

[9] Highsmith J., Orr K. Adaptive Software Development. A Collaborative Approach to Managing Complex Systems, *Dorset House* (2000).

[10] Feature Driven Development, http://www.feature drivendevelopment.com.

[11] Beck, K. Extreme Programming Explained. Embrace Change, *Pearson Education*, 1999.

[12] Berka, K., Measurement: Its Concepts, Theories and Problems (Boston Studies in the Philosophy of Science), *Kluwer,* Vol. 72 (1982).

[13] Fenton, N, Pfleeger, S.L., Software Metrics: A Rigorous and Practical Approach, 2nd Edition, *PWS Publishing Co*, EEUU, ISBN 0534954251 (1998).

[14] Abrahansson, P., Salo, O., Ronkainen, J.,Warsta, J., Agile Software Development Methods, Review and Analysis, *VTT Publications,* 478 (2002).

[15] Iacovelli, A., Souveyet, C., Framework for Agile Methods Classification, *Workshop on Model Driven Information Systems Engineering: Enterprise, User and System Models* (2008).

[16] Strode, D.E., The Agile Methods: An Analytical Comparison of Five Agile Methods and an Investigation of Their Target Environment, MSc Thesis in Information Systems, *Massey University*, Palmerstin North, Nueva Zelanda (2005).

[17] Visconti, M., Cook, C., An Ideal Process Model for Agile Methods, *LNCS*, ISBN 978-3-540-21421-2, Vol 3009, pp.439-441 (2004).

[18] Tsun Chow, Dac-Buu Cao, A Survey Study of Critical Success Factors in Agile Software Projects, *School of Business and Technology, Capella University*, Minneapolis, MN 55402, USA (2007).

[19] Schatz, B., Abdelshafi, I., 2005. Primavera Gets Agile: A Successful Transition to Agile Development. *IEEE Software 22*.

[20] Karlstrom, D., Runeson, P., 2005. Combining Agile Methods with StarGate Project Management. *IEEE Software.*

[21] Iacovelli, A.: Introduction de l'Agilit´e dans les Methodes. Master thesis, *University Paris 1 Pantheon Sorbonne* (2007).