Framework for Agile Methods Classification

Adrian Iacovelli and Carine Souveyet

Centre de Recherche en Informatique (CRI),
Université Paris 1 - Panthon Sorbonne,
90 rue Tolbiac, 75013 Paris
{adrian.iacovelli,carine.souveyet}@univ-paris1.fr

Abstract. Agile methods are the response to turbulent software development environments and requirements definitions differs in these methods from what is done in others. The purpose of this paper is to classify these former methods to measure their impact on requirement engineering processes. The approach used in this research has several purposes, the first one is to build a framework to classify and compare the methods. The second is to propose a component based approach to bring agility to other methods.

Key words: Agile methods, Method engineering

1 Introduction

In the mid 90's people start creating new methods because industrial requirements and technologies are moving fast and customers were unable to define they needs in early stages of the projects [1, 2]. These new methods, called the agile methods are designed to respond to disruptive software environments where requirements are constantly changing.

Requirements in agile methods are quite different from what is done in classical plan-driven methodologies. In the latter, requirements are elicited at the beginning of the project and suppose to remain the same all along the project[3], whereas in the former, requirements changed constantly. As long as the project is going on, the requirements definition is detailed. They become more and more precise at each iteration and changes are integrated through the process. So as part of an agile process the requirements definition is iterative and incremental.

The scope of this paper is to classify agile methods to compare them and evaluate their impact on requirement engineering processes. A framework is proposed in section 2 and applied to eight agile methods in section 3 for classify them. Finally an approach of agile methods reusable components will be proposed in section 4.

2 Classification Framework

2.1 Framework Introduction

The framework for classifying agile methods is based on a faceted approach similar to the one used in [4, 5]. The aim is to classify methods through four views,

each one representing an aspect of the methodologies. Each view is characterized by a set of attributes.

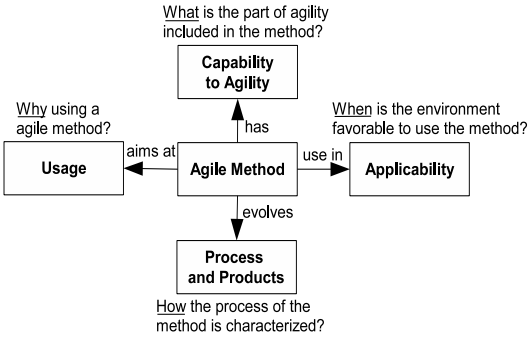

Fig. 1. The four views of the agile methods.

As shown in Figure 1, the four views are the decomposition of an agile method. They capture why and when using agility. In other terms what are benefits to objectives brought by the method agility and what's the favourable environment to its application. These aspects are correlated to what is the method agility and how this agility is expressed in practice. This means what are the parts of agility concept supported by method guidelines and rules to satisfy objectives mentioned below. Added to how these rules are derived in activities and products.

2.2 Usage View

This view captures why using the agile method. The attributes of the view aim to evaluate all the benefits that the development team and the customer can gain by applying the method.

According to the quantitative approach used in [6], applying agile methods can increase a productivity, quality and satisfaction. The methods can provide guidelines to increase benefits such as productivity gain, end user satisfaction and quality level respect attributes of the usage view.

Behind the agility concept some aspects can be identified as the integration of changes in the development process. So agile methods are providing rules and guidelines to fit turbulent environments and providing the respect of requirements and delivery dates in such environments. These three last aspects will be integrated in this view as attributes.

A study on the limitations of agile methods [7] points out that they provide a limited support to subcontracting, agile methods can bring some flexibility to outsourcing contracts. Defining contracts in two parts, a static one and a variable one creates this flexibility. So let's carry our interest if some agile rules can be

favourable to off shoring by integrating the favourable to off shoring attribute in this view.

The attributes of the usage view are :
Adapted to turbulent environments : BOOLEAN.
End user satisfaction : BOOLEAN.
Favourable to off shoring : BOOLEAN.
Productivity gain : BOOLEAN.
Respect of a quality level : BOOLEAN.
Respect of delivering dates : BOOLEAN.
Respect of the requirements : BOOLEAN.

2.3 Capability to Agility View

The capability to agility view represents what is the part of agility in the method, how agile is the method. The attributes of this view will represent all aspects of the agility concept and their valuation will reflect what are the aspects included in the method.

A software development method is composed by a life cycle, specifics concepts about the development itself and the organization of the people around the method. Let's start with the first one, the life cycle. In software engineering various life cycle models exits, for example V model [8], waterfall model [9] or spiral model [10]. According to the chronology of agile methods apparition related in [11], most of these methods are directly derived from spiral model. This is explained by the two main characteristics of the spiral life cycle, an iterative and incremental behaviour. Such life cycle provide a development of the software increment by increment. So environmental and requirements changes can be integrated to every iteration of the process and the work plan would not be fixed, it will change through iterations. Another point of interest of this behaviour is the length of iterations. Shorter iterations will increase the number of meetings with the customer to define and detail their needs incrementally bringing reactivity to the method. From observations, six attributes can be identified : short iterations, reactivity, integration of the changes, changes of the functional requirements, changes of the non functional requirements, change of the work plan.

Concerning the organization of people, agile methods tend to break contractual relations between customers and development teams [12]. This relation will be expressed in this view by the collaborative attribute. The organization aspect also concerns human relations into the development team. An agile team is a kind of holographic organization where each member has the knowledge of the whole system [3]. So if a member left the team no knowledge is lost. Furthermore people are in the central place of the method and some decisional power is delegated to development teams. These last two aspects are declined in the view by the people centred, human resources can change and knowledge sharing attributes.

Some specific concepts of agility are included by the agile method in activities of software development itself. The major concept is light weight of the process. This concept is expressed in the agile manifesto [1] and the first name

of agile methods was light weight methods. Globally, agile methods include less documentation and modeling tasks than plan-driven approaches. Two other concepts concern the code testing and refactoring. Testing is a strong practice of agility, tests grant you that the software you are developing corresponds to the customer requirements and that your code is not regressing by the introduction of errors in functionalities previously done. The non regression aspect goes hand in hand with the refactoring one which you are always reflecting on simplifying and improving the quality of the code [2]. These three concepts will be declined in attributes in this view. Another one which is the change indicators attribute will also be added to this view and it represents if the method as some metrics for helping to introduce the changes through the process.

The attributes of the Capability to Agility View are :
Change indicators : BOOLEAN.
Collaborative : BOOLEAN.
Functional requirements can change : BOOLEAN.
Human resources can change : BOOLEAN.
Integration of the changes : BOOLEAN.
Knowledge sharing : ENUM(low, high).
Light weighted : BOOLEAN.
Non functional requirement can change : BOOLEAN.
People centred : BOOLEAN.
Reactivity : ENUM(at the beginning of the project, each milestone, each iteration).
Refactoring politic : BOOLEAN.
Short iterations : BOOLEAN.
Testing politic : BOOLEAN.
Work plan can change : BOOLEAN.

2.4 Applicability View

The aim of this view is to show the impact of environmental aspects on the method. It represents when the environment is favourable to apply the agile method. This aspect will be described by attributes, each one corresponding to a characteristic of the environment.

Software development environments can be characterized by indicators. A previous research [13] has determined a metric measuring the fitness of a project environment with the agility concept to determine which software development method use. This metric is called the Agility Measurement Index (AMI). It will not be used itself in this framework but our interest will be on the environmental projects indicators constituting the AMI (duration, risks, novelty, effort and interaction dimensions). These indicators will be derived in attributes of this view to characterize an ideal project environment to apply the agile method.

The duration represents the deadline of the project and will be expressed by the project size attribute. The risks are the degree of criticality of the software, for example a software impacting or monitoring high economic issue for the customer is highly critical. This aspect will correspond to the project risks attribute.

Effort indicator of the AMI represents the number of person-hour provided in the project by the customer and the development team. I think this aspect is not pertinent in this form to characterize an environment because effort is the duration of the project combined with its team size. So the team size will be an attribute in this view and be more relevant to identify a favourable environment for applying the method. The AMI novelty indicator represents the ability for the project to integrate a novelty solution and will be deported in the integration degree of novelty attribute. The last indicator is the interaction dimensions and corresponds to degree of interactions between the customer and the development team. As seen previously the people organizational aspect is important in agile methods. To reflect this, the interactions aspect of the AMI will be extended to the interactions between end users and development teams. It will also concern interactions and organisation between the development team members. These aspects will be derived into attributes. A last aspect will be added to the applicability view, the project complexity attribute.

The attributes of the Applicability View View are :
Degree of interaction between the team members : ENUM(low, high).
Degree of interaction with the customer : ENUM(low, high).
Degree of interaction with the end users : ENUM(low, high).
Degree of novelty integration : ENUM(low, high).
Project complexity : ENUM(low, high).
Project risks : ENUM(low, high).
Project size : ENUM(small, large).
Team organization : ENUM(self organization, hierarchical organization).
Team size : ENUM(small, large).

2.5 Process and Products View

The process and product view represents how is characterised the method and what are the products of activities of the method process. The attributes will characterised the agile method process by two dimensions and list the products of the process activities.

A previous research [12] has a part of its agile methods comparison done on their life cycle. The methodology process is composed of two dimensions. The first dimension is the software development activities covered by the agile method. The second one represents the method abstraction level of its guidelines and rules. These two dimensions will be captured by attributes of this view and we will also carry our interest on the products of the process activities as a third attribute for the process and products view.

The attributes of the Process and Products View are :

Abstraction level of the rules and guidelines : SET(ENUM(project management, process description, concrete rules and guidelines on activities and products)).

Activities covered by the agile method : SET(ENUM(launching of the project, requirements definition, modeling, code, unit test, integration test, system test, acceptation test, quality control, system use)).

Products of the method activities : SET(ENUM(design models, commented source code, executable, unit tests, integration tests, system tests, acceptation tests, quality reports, user documentation)).

3 Application of the Framework

The framework presented previously will be applied to eight major agile methods in this part. Methods are: Adaptive Software Development (ASD) [14], Agile Modeling (AM) [15], Crystal Methodologies [16], Dynamic System Development Method (DSDM) [17], Extreme Programming (XP) [18], Feature Driven Development (FDD) [19], Pragmatic Programming (PP) [20] and Scrum [21]. Each method will be characterized by the framework view and attributes according to their descriptions. The rationale of the methods evalution is fully documented in [22]

Legend for the process and products view :
- "Activities" attributes :
launching of the project = l, requirements definition = rd, modeling = m, code = c, unit test = ut, integration test = it, system test = st, acceptation test = at, quality control = qc, system use = su
- "Abstraction level" attributes :
project management = pm, process description = pd, concrete rules and guidelines on activities and products = crg
- "Products" attributes :
design models = dm, commented source code = csc, executable = exe, unit tests = ut, integration tests = it, system tests st, acceptation tests = at, quality reports = qr, user documentation = doc

From this Table 1 we can identify that some methods have particular characteristics in comparison of the others. For example DSDM is the only method integrating a launching of the project activity. When going deeper in analyse of this comparison, we can notice that some attributes are common to several agile methods. From these common characteristics, we can regroup the methods into relevant sets of methods.

The most numerous common attributes of agile methods allow to capture them into two main classes as represented in Figure 2. The first class is regrouping the AM, XP and PP methods. These methods are characterized by a light weighted process with short iterations. They are addressed to small projects and teams with low risks. They also provide a productivity gain when they are applied in these conditions. According to this set of common attributes, this class represents methods oriented on software development practices. These methods are composed of rules and guidelines on the development activity itself. They concentrate the efforts on how increase integration of changes, correctness, quality and productivity of software.

The second class of method regroups the ASD, Crystal, DSDM and Scrum methods by having the same level of abstraction of their rules (project man-

	ASD	AM	Crystal	DSDM	XP	FDD	PP	Scrum
Usage view								
Respect of delivering dates	True	False	True	True	False	True	True	True
Respect of the requirements	True	True	True	True	True	True	False	True
Respect of a quality level	True	False	False	False	False	False	True	False
End user satisfaction	False	False	False	True	False	False	False	False
Turbulent environment	False	True	False	True	True	False	True	True
Favourable to off shoring	True	False	True	True	False	False	False	True
Productivity gain	False	True	False	False	True	False	True	False
Capability to agility view								
Short iterations	False	True	False	False	True	True	True	True
Collaborative	True	True	True	True	True	False	True	False
People centred	True	True	True	True	True	False	False	False
Refactoring politic	False	True	False	True	True	False	True	False
Testing politic	True	False	False	True	True	False	True	True
Integration of the changes	True	True	False	True	True	False	True	True
Light weighted	False	True	False	False	True	True	True	True
FR can change	True	True	False	True	True	False	True	True
NFR can change	True	False	False	False	False	False	False	False
Work plan can change	False	True	False	False	True	False	True	False
Human resources can change	True	True	True	False	True	False	False	False
Change indicators	False	False	False	False	True	False	False	False
Reactivity	Milestone	Iteration	Milestone	Iteration	Iteration	Project beginning	Iteration	Milestone
Knowledge sharing	High	High	High	Low	High	Low	Low	Low
Project size	Large	Small	Large	Large	Small	Large	Small	Large
Project complexity	High	Low	High	High	Low	High	Low	High
Project risks	High	Low	High	High	Low	High	Low	High
Team size	Large	Small	Small	Large	Small	Large	Small	Small
Interactions with customers	High	High	Low	High	High	Low	Low	Low
Interaction with end users	Low	Low	Low	High	Low	Low	Low	Low
Team members interactions	Low	High	High	High	High	Low	High	Low
Degree of novelty integration	High	Low	Low	High	High	Low	Low	Low
Team organization	Hierarchical	Self	Self	Hierarchical	Self	Hierarchical	Hierarchical	Self
Process and products view								
Activities covered by the method	rd, m, c, ut, it, st, at, qc	rd, m, c	m, cut, it, st	l, rd, m, c, ut, it, st, at, su	rd, m, c, ut, it, st	rd, m, c, ut, it, st, qc	rd, m, c, ut, it, st	rd, m, c, ut, it, st
Abstraction level of the guidelines	pm, pd	crg	pm, pd	pm, pd	pd, crg	pm, pd, crg	crg	pm
Products of the method activities	csc, exe, ut, it, st, st, at, qr	dm, csc, exe	csc, exe, ut, it, st	dm, csc, exe, ut, it, at, doc	csc, exe, ut, it, st	dm, csc, exe, ut, it, st	dm, csc, exe, ut, it, st	dm, csc, exe, ut, it, st

Table 1. Agile methods comparison

agement). These methods also distinguish from the others by their respect of requirements and delivering dates. Another common point to them is that they adapted to large and complex project. From this common set of attributes it emerges the class of project management oriented methods. These methods are oriented on the management of the project life cycle to fit large projects.

One method, FDD differentiates from the others by owning characteristics of the two classes. FDD is a hybrid method inheriting the development practices from the first class and some project management guidelines from the second. This is captured in Figure 2 by an hybrid class which inherits characteristics of the two other main classes.

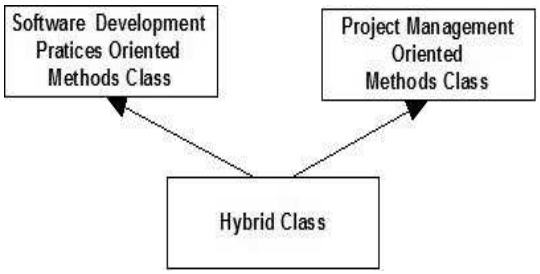

Fig. 2. Agile Methods Classes.

All agile methods are regrouped into these three classes :

Software Development Practices Oriented Methods : Agile Modeling, Extreme Programming, Pragmatic Programming.

Project Management Oriented Methods : Adaptive Software Development, Crystal Methodologies, Dynamic System Development Method, Scrum.

Hybrid Methods : Feature Driven Development.

Some other minor classes, shown in Table 2 below, can be captured by the same means. They reflect common particular aspects of agile methods. For example if we carry our interest on the quality in agile methods. We can find a quality control class composed by FDD and ASD. This subclass represents that the process contains activities for controlling quality of the software to guaranty a quality level of it. In the same way we can identify two other subclasses, the knowledge management and the high reactivity subclasses. The first one concerns high interactions with customers and the team members. Knowledge management is also identified by self organisation of the teams providing a high sharing of the knowledge. These factors impact humans resources, when a member leave the team the other member have already integrated his knowledge. In practice the knowledge management can be found in some guidelines, for example the pair programming in Extreme Programming providing the share of knowledge on the whole system between the team members. The last subclass represents the speed of integration of changes in the process. This is characterised by a high reactivity with a meeting between the customer and the team every iteration.

This high degree of interaction with the customer and the team reflect the fitness of the method to turbulent environments when the customer's requirement are frequently changing.

	Quality Control	**Knowledge Management**	**High Reactivity**
AM		X	X
ASD	X		
Crystal		X	
DSDM			X
FDD	X		
PP			
SCRUM			
XP		X	X

Table 2. Discriminant Characteristics for Agile Methods

This Table 2 captures the minor classes and shows which methods are regrouped in these classes.

This work leads to classify agile methods and provides a support to choose the right method according to the project context. The second topic of the paper is to analyse agile methods for extracting best practices which can be reused in plan-driven methodologies. Section 4 deals with this topic and explains how the framework is used to extract agile method components.

4 Extracting Agile method Components from the framework

The purpose of this section is to identify best practices of agile methods which can be reused in plan-driven methods. Such components can be used to improve the agility of a method. We exploit the framework proposed in section 2 to indentify such components. The four aspects what, how, when, and why is unsefull to indentify relevant best practices. what and how asects allow to capture the best practices. Their relevance is provided by the analysis by the when and why aspects. This approach has been applied for agiles characteristics common to most of the methods and those specifics to few methods. This leads to identify eight components.

If we carry our attention on DSDM for example, two components can be captured. The first one concerns the "launching of the project" activity expressed by a feasibility and a business study. This component is captured in the framework by the presence of launching of project activity in the method life cycle. It aims to estimate if the project is feasible for the requirements and the dates announced. It can be applied in large and complex project environments. The second component concerns the "management of the end users" by integrating the system users in the development process, giving them some decisional power on the requirements for the system features. It also included the validation of the deliverables and the formation of end users to the new system. This component

is captured in the framework through the presence of activities for the system in use in the life cycle, the production of user documentation, high level of interactions with end users and the aim to satisfy users of the system. This last attribute reflects the aim of this component to increase the satisfaction of the users of system and reduce their aversion to novelty. From another method like ASD, we can extract a "quality" component concerning the activities of quality controlling and the integration of non functional requirements changes in the process. This component is isolated from the quality level respect, integration of the non functional requirement, activities and products of quality control framework attributes. The aim is to provide a quality level for developed software along with an increase of the productivity. This quality gain can by applied in complex and risky projects with large development team and for controlling the development of the novelties.

Now let's carry our interests on the aspects issued from the agility concept that can be captured into components to bring agility to other methods. On the aspect of agility concerning the code we can identify two components. The first one is the "tests" and concern test politics, testing activities and products. This component can be applied in every project and aims to produce a productivity gain. The second component is "refactoring" by reviewing the code constantly to simplify and improve it. This is destined to small projects with low complexity and risk to gain productivity. In a more general consideration "knowledge management" can be captured into a component to satisfy the same objectives in projects with small teams and a high level of interaction into them. This component is materialised by the sharing of the knowledge on the system between the teams members. Concerning the framework, this component is issued from the following attributes : high interactions between the team members, self organisation for the teams, high amount of knowledge sharing and human resources can change.

	What	**How**	**Why**	**When**
Launching of the Project		Feasibility and business study	Respect of delivering dates	Large and complex projects
Management of the End Users	People centred and Reactivity	Involving users in the process activities	Increase end users satisfaction	Projects with a high interaction rate
Quality	Changes in NFR	Quality control activities and products	Respect of a quality level	Complex and risky projects
Tests	Test politic	Testing activities and products	Productivity gain	All projects
Refactoring	Refactoring politic	Constant code review	Productivity gain	Project with low risks and complexity
Knowledge Management	People centred and knowledge sharing	Self organizing Teams and high interaction rate	Productivity gain	Small teams with high interaction rate
Agile Life Cycle	Iterative and incremental life cycle	Short iterations and meetings with the customer	Fit turbulent environments	High interaction rate
Change Indicator	Manage the Changes	Project velocity metric	Respect of delivering dates	All projects

From a requirement engineering point of view two relevant components can be captured. One about the "agile life cycle" and another on a "change indicator". An agile life cycle is iterative and incremental and an agile requirement definition follows this principle. The customer defines his needs in a high level of abstraction at the beginning of the project. Every iteration, requirements definition will be detailed at a meeting between the customer and the development team. Only the needed features implemented in the current iteration will be defined in detail. This prevents from the changes on initial requirements and integrates customers feedback in the definition. From the framework, attributes characterising an optimum agile cycle are short iterations and adapted to turbulent environment. Such component applies in projects where a high cooperation with the customer is possible and aims to integrate changes in requirements happening in turbulent environments.

Change indicators can be used in development process to control and manage the changes. In this component we will carry our interest on project velocity used in XP. It's a metric allowing customer and development teams to refine their time to code estimations on features to develop in the current iteration. The project velocity is the factor between estimations in ideals programming days and real time it takes to develop. It's calculated from the sum of estimations for the previous iteration divided by the sum of the real time it takes to implement. So it considers also current team productivity to refine the estimations. This component aims to improve the respect of delivering dates by helping customer and development teams to make betters estimations on implementations of requirements and applies in every kind of agile project.

The list of identified components is not exhaustive.

Table 4 shows a summarization of components characteristics according to aspects expressed in the framework views.

5 Conclusion

The paper describes a framework for agile methods used in two different topics:

- Classify agile methods to support the method selection.
- Improve agility of plan-driven methods by proposing agile components.

The first step was to define a framework to describe agile methods. Ones applied to agile methods, a classification has been done by regrouping methods common attributes. From this, several methods components have been captured to be reused into other methods.

The agile method components can be used in a component based method engineering aproach to improve agility of existing methods. In fact, they can be used to adapt methods to turbulent environments or to upgrade agile methods with bringing news aspects for a particular kind of projects. This work is a preliminary work to the definition of a method engineering approach aiming at increasing the agility of methods.

References

1. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., Thom, D.: Manifesto for agile software development. Website (2001) `http://agilemanifesto.org/`.
2. Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F., Tesoriero, R., Williams, L., Zelkowitz, M.: Empirical findings in agile methods. In: Agile Universe. (2002) 197–207
3. Nerur, S., Balijepally, V.: Theoretical reflections on agile development methodologies. Communications of the ACM **50**(3) (2007) 79–83
4. Rolland, C.: A comprehensive view of process engineering. In Pernici, B., Thanos, C., eds.: 10th International Conference on Advanced Information System Engineering, CAISE'98, Springer-Verlag (1998)
5. Rolland, C., Achour, C.B., Cauvet, C., Ralyt, J., Sutcliffe, A., Maiden, N., Jarke, M., Haumer, P., Pohl, K., Dubois, E., Heymans, P.: A proposal for a scenario classification framework. Requirement Engineering (1998) 11–26
6. Parsons, D., Ryu, H., Lal, R.: The impact of methods and techniques on outcomes from agile software development projects. International Federation for Information Processing **30** (2007) 11–26
7. Turk, D., France, R., Rumpe, B.: Limitations of agile software processes. In: International Conference on eXtreme Programming and Agile Processes in Software Engineering. (2002)
8. Rook, P.: Controlling software projects. Software Engineering Journal **1**(1) (1996) 7–16
9. Royce, W.W.: Managing the development of large software systems. (1987)
10. Boehm, B.W.: A spiral model of software development and enhancement. IEEE Computer **21**(5) (1988) 61–72
11. Abrahamssona, P., Warstab, J., Siponenb, M.T., Ronkainena, J.: New directions on agile methods: A comparative analysis. In: International Conference on Software Engineering. (2003)
12. Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J.: Agile software development methods : Review and analysis. VTT Publication (2002)
13. Datta, S.: Agility measurement index - a metric for the crossroads of software development methodologies. ACM (2006) 271–273
14. Highsmith, J.A.: Adaptive Software Development : A Collaborative Approach to Managing Complex Systems. Dorcet House Publishing (2000)
15. Ambler, S.W.: Agile modeling. Website (2006) `http://www.agilemodeling.com`.
16. Cockburn, A.: Agile Software Development. (2001)
17. Consortium, D.: Dsdm version 4.2. Website (2007) `http://www.dsdm.org/version4/2/public/default.asp`.
18. Wells, D.: Extreme programming. Website (2006) `http://www.extremeprogramming.org`.
19. Palmer, S.R., Felsing, J.M.: A Practical Guide to Feature-Driven Development. (2002)
20. Hunt, A., Thomas, D.: Pragmatic Programmer, The: From Journeyman to Master. (1999)
21. Schwaber, K.: Agile Project Management With Scrum. (2004)
22. Iacovelli, A.: Introduction de l'agilité dans les méthodes. Master thesis, University Paris 1 Panthéon Sorbonne (2007)