2010 Seventh International Conference on the Quality of Information and Communications Technology

Classification and Comparison of Agile Methods

João M. Fernandes Mauro Almeida
Dep. Informática / CCTC, Universidade do Minho, Braga, Portugal
jmf@di.uminho.pt, mauro.almeida@gmail.com

Abstract

This manuscript describes a technique and its tool support to perform comparisons on agile methods, based on a set of relevant features and attributes. This set includes attributes related to four IEEE's Software Engineering Body of Knowledge (SWEBOK) Knowledge Areas (KAs) and to the agile principles defined in the Agile Manifesto. With this set of attributes, by analysing the practices proposed by each method, we are able to assess (1) the coverage degree for the considered KAs and (2) the agility degree. In this manuscript, the application of the technique is exemplified in comparing eXtreme Programming (XP) and Scrum.

1. Introduction

This manuscript presents a technique to compare and classify agile methods, using as criteria a set of selected attributes. The technique intends to be a contribution for the creation of a guide to help developers on the selection of the software development method (focusing on agile methods) that best fits a given development context. The attributes chosen for this study were selected to assess, with respect to each method, the coverage degree to four SWEBOK KAs (that theoretically are transversal to all development methods) and the agility degree.

The proposed technique is exemplified by comparing two agile methods: XP and Scrum. This choice is due to two main reasons: (1) those agile methods are among the most cited in literature and (2) the increasing number of development teams that currently are adopting or are considering adopting one of these two methods.

We also point out the similarities and the differences between both methods. Based on this analysis, we conclude that each method explicitly defines a set of practices with distinct focuses. For example, XP is predominantly concentrated on the implementation and test phases of a software project, while Scrum was conceived essentially to support the management of the projects. In addition, this manuscript presents a worksheet that generates a report based on the conducted study and on the inputs from the users.

The Agile Software Solution Framework (ASSF), a complete framework to assist in the assessment of an enterprise necessary agility degree and to identify the appropriate way of introducing agility into the organisation is described in [1]. The major element of this framework is the Agile Toolkit, which provides an embedded analytic tool that allows the comparison of agile methods (4-DAT). The 4-DAT approach [2] examines software methods from four perspectives: (1) method scope, (2) agility characterisation, (3) characterisation of agile values, based on the ones proposed in the Agile Manifesto, and (4) software process characterisation. These perspectives were defined based in different studies and distilled from key aspects of agile methods. This differs from our approach which uses five perspectives based on the SWEBOK defined KAs and the Agile Manifesto defined values and practices. Additionally, the perspectives presented in [2] are partly qualitative and partly quantitative. Our approach adopts qualitative analyses which are complemented with tool support that uses a quantitative parametric scale, adjusted by the user.

A proposal to assess the suitability of a software project to the adoption of an agile method is presented in [3]. This assessment is based on ten *Critical Adoption Factors*. This approach differs from ours, since we do not try to assess the suitability of a software project to a given agile method, but rather determine the 'best' method for a project.

This manuscript has the following structure. In section 2, the main characteristics of agile methods and the Agile Manifesto Principles are presented. A brief description of XP and Scrum is also included. The technique proposed in this work to classify and compare agile methods is explained in section 3. Section 4 is the central part of the manuscript and presents the classification and comparison of XP and Scrum, based on the proposed analysis method. Conclusions and future work are discussed in section 5.

2. Agile Methods

Agile methods are based on the notion of incremental development, a technique introduced in 1975 [4]. In 2001, enthusiasts of these new development methods came together to discuss this issue. They proposed the Agile Manifesto (*http://agilemanifesto.org/*), that defines the following values, which are the cornerstones of the agile methods:

- **Individuals and interactions** over processes and tools;
- **Working software** over comprehensive documentation;
- **Customer collaboration** over contract negotiation;
- **Responding to change** over following a plan.

978-0-7695-4241-6/10 $26.00 © 2010 IEEE
DOI 10.1109/QUATIC.2010.71

Additionally, twelve principles that support the values presented above were defined: (1) The highest priority is to satisfy the customer through early and continuous delivery of valuable software; (2) Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage; (3) Deliver working software frequently, from a couple of weeks to a couple of months; (4) Business people and developers must work together daily throughout the project; (5) Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done; (6) The most efficient and effective method of conveying information to and within a development team is with face-to-face conversation; (7) Working software is the primary measure of progress; (8) Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace; (9) Continuous attention to technical excellence and good design enhances agility; (10) Simplicity — the art of maximizing the amount of work not done — is essential; (11) The best architectures, the best requirements, and the best designs emerge from self-organizing teams; (12) The team regularly reflects on how to become more effective.

These principles have two main objectives: (1) to promote a better understanding of what agile methods are, and (2) to guide the project teams to determine if they are in fact using an agile method. This manifesto, and all its values and principles, represent the philosophy behind agile methods and ideally should be present in all practices proposed by the various agile methods.

In 1990, Beck et al. concluded that communication, simplicity, frequent feedback and courage are the main values for improving the process of software development [5]. Based on these values, XP was proposed in 1999 [6] to mitigate the problems caused by long periods of development without feedback from the client that were until then (and still are) common in many software projects. To attain this aim, XP proposes a set of 12 practices, described in table 1, that should be used in software projects [6], [7].

Sutherland proposed in 1993, based on the ideas and concepts presented by Takeuchi and Nonaka in 1986 [8], a new development method, which was, two years later, formalized and adapted for software development under the name of Scrum [9]. The use of Scrum has already been attempted in projects with a higher dimension than the one which agile methods are usually applied to [10].

As XP, Scrum also proposes a set of practices with the objective of facilitating and promoting efficient software construction (table 2).

3. Proposed Technique

Conceiving a technique to perform comparisons on software development methods constitutes a great challenge.

Practice	***Description***
The Planning Game	This practice suggests a close relationship between the client and the technical team of the project. Each part is responsible for defining and identifying a set of project-specific attributes, such as scope, deadlines, effort estimate and technologic drawbacks.
Small Releases	The objective of this practice is to quickly put in production (i.e., to implement and test) each new small set of features. Each release should be as small as possible and contain the most valuable requirements for the client.
Metaphors	Metaphors allow to describe a feature to be implemented, by creating a common vision of the client and the technical team on how the product should work. This way one can reduce the use of technical expressions, often difficult to be understood by the client.
Simple Design	The architecture and the code (including the unit tests) should be as simple as possible.
Test-Driven Development	All implemented features must be covered by unit tests, which must all be always satisfied, in an effort to eliminate unit-level and regression bugs during development. In XP, a feature is only ready to be integrated in a release when it meets these requirements.
Refactoring	Refactoring aims to simplify implemented code by removing code ambiguity and redundancy.
Pair Programming	This practice consists in having two programmers working simultaneously in the same computer. Each programmer has a specific role. While one element is responsible for writing the code, the other is responsible for verifying and validating it, with particular attention to untested features or code blockage.
Collective Code Ownership	Each team member is encouraged to perform all necessary changes in the code. Thus, all team members are owners of the code. This practice avoids unnecessary waits for third party changes into the code.
Continuous Integration	After a new feature is implemented or the code is adjusted, and all tests are successfully executed, a new release should be created reflecting all the changes.
40-Hour Weeks	No element of the team should work a second consecutive week of overtime [7]. If so, the problem should be addressed as a project management issue.
On-site Customer	XP proposes not only a close relationship with the client, but also that a client (or a representative) should always be present during the life-cycle of the project, therefore being part of the team of the project.
Coding Standards	Coding standards allow a easier interpretation of the implemented code by all programmers.

Table 1: Proposed practices in XP

The main difficulties that contribute to the essence of the challenge are: (1) Similar application contexts for the considered methods, both before and after their application, are hard to find; (2) Scientific evidences of the proposed ideas are difficult to provide; (3) The experience and intuitions of the persons conducting the study need to be considered.

To reduce and overcome these difficulties, we adopt the *quasiformal* comparison presented in [11]. Five different actions for the *quasiformal* comparison are suggested and we have chosen to use the second alternative, which implies that we need to distill a set of important features inductively from several methods and compare each method against it.

Regarding the attributes used in our technique, we selected four of the eleven knowledge areas (KAs) defined in the SWEBOK [13]: (1) Software Requirements, (2) Software Construction, (3) Software Testing, and (4) Software Engineering Management. With this subset of attributes, we intend to assess the coverage degree of each method with respect to the selected KAs. Although covering the full set of the SWEBOK KAs is obviously the best option and a priority in future work, the present study consists on an exploratory approach focusing on proving the concepts presented. Therefore, the selected subset is enough given the scope of the study. Additionally, we believe that this subset is broad enough to include the critical activities in any software development context. A fifth attribute, which relates the agile principles of the Agile Manifesto and the practices advocated by a given agile method, was selected in order to assess the agility degree of the methods. Table 3 summarises the selected attributes.

4. Classifying and Comparing XP and SCRUM

To classify the existence of practices, advocated by the agile methods, that support the selected KAs and to characterise the coverage of the principles of the Agile Manifesto by each method, we use the criteria presented in table 4.

Practice	*Description*
Product Backlog	Represents the list of all features and requirements to be implemented. The product owner is responsible for creating and maintaining this list. A priority and estimated effort is associated to each item of the Product Backlog.
Effort Estimation	Effort estimation is the iterative process through which the effort to perform an item of the Product Backlog is predicted.
Sprint	Sprints are considered the agile part of Scrum and they consist on a development cycle through which the Scrum team organizes itself to produce a new increment to be integrated into the new version of the product, completed at the end of each sprint. Each sprint usually lasts from 2 to 4 weeks [12].
Daily Meeting	Meetings are held daily, lasting 15 minutes or less. These meetings aim to analyze the progress of the project and the unexpected issues that may delay the project, by identifying the work undertaken since the last meeting and the work to be performed until the next one.
Sprint Planning Meeting	The sprint planning meeting is divided into two parts. During the first part of the meeting, the Scrum master, the client, the product owner and the Scrum team select a set of items of the Product Backlog to be implemented during the sprint. Next, the selected items are decomposed into tasks to be performed by the Scrum team elements. These tasks will integrate the sprint backlog.
Sprint Backlog	Represents the set of activities to be performed during a sprint. Each activity or set of activities represents an item of the Product Backlog that should be integrated in the next release, to be available at the end of the current sprint.
Sprint Review Meeting	At the end of each sprint, a review meeting is held. This meeting is attended by all stakeholders and a demonstration of all the features of the new release is performed.
Sprint Retrospective	Its purpose is to promote discussion among the project team elements about issues to which the team may have been exposed to and how these can be tackled in future sprints.
Sprint Burn Down Chart	The Sprint burn down chart is a publicly displayed chart showing the remaining work of the sprint backlog, updated every day, giving a simple view of the work to be performed until the end of the current sprint.

Table 2: Proposed practices in Scrum

The challenge in quantifying the coverage of a given sub-attribute or principle, by a set of practices proposed by each analysed method, has lead to the choice of a qualitative classification system. Although simple, a qualitative classification system satisfies the objective of our technique. In the remainder of this manuscript, due to space limitations, we only show the results obtained for two attributes (Software Requirements and Software Construction), although similar results were obtained for the other attributes.

Attribute	*Description, sub-attributes and principles*
Software Requirements	Identifies how the method addresses the requirements analysis in a software project, particularly regarding the following sub-attributes: • Software Requirements Fundamentals; • Requirements Process; • Requirements Elicitation; • Requirements Analysis; • Requirements Validation;
Software Construction	Identifies how the method deals with the software implementation, particularly regarding the following sub-attributes: • Minimizing Complexity; • Response Capability to Unexpected Changes; • Construction for Verification; • Standards in Construction.
Software Testing	Identifies how the method validates the implemented features and the approach adopted for testing, particularly regarding the following sub-attributes: • Software Testing Fundamentals; • Test Levels; • Test Related Measures; • Test Process.
Software Engineering Management	Identifies how the method addresses the project management, particularly regarding the following sub-attributes: • Initiation and Scope Definition; • Software Project Planning; • Software Project Enactment; • Closure; • Review and Evaluation.
Agile Principles vs. Proposed Practices	Identifies how the principles presented in the Agile Manifesto relate with the practices introduced by the agile method.

Table 3: Set of attributes and sub-attributes used for comparing agile methods

NS	Not Satisfied	None of the proposed practices or concepts of the method support the sub-attribute or principle
PS	Partially Satisfied	The proposed practices or concepts of the method support the sub-attribute or principle, although some of its aspects are not considered
AS	Adequately Satisfied	The proposed practices or concepts of the method entirely support the sub-attribute or principle

Table 4: Classification criteria adopted in the comparative analysis

4.1. Software Requirements

Concerning the Software Requirements attribute, each agile method was compared against all the sub-attributes presented in table 5.

4.1.1. XP. XP proposes a set of practices and concepts that support the definition, the elicitation and the analysis of the requirements (i.e., Planning Game, exploration phase of the life cycle, and definition of the user stories). However, XP does not clearly and objectively distinguishes the different types of existing requirements, therefore we consider that it **partially satisfies** the sub-attribute *1. Software Requirements Fundamentals*.

Sub-attribute	*Description*
1. Software Requirements Fundamentals	Refers to the clear definition of software requirements, distinguishing between different types of requirements (e.g., product vs. system, functional vs. non-functional)
2. Requirements Process	Refers to the definition of a clear process to the specification, the analysis and the validation of requirements, specifying all actors, proposed practices, and management models
3. Requirements Elicitation	Refers to the way software requirements are elicited and which actors are involved in the process
4. Requirements Analysis	Refers to the proposed practices for detecting and solving conflicts and for classifying software requirements, according to a predefined metric
5. Requirements Validation	Refers to the proposal of concepts, practices or techniques to allow the validation of the implemented requirements

Table 5: The sub-attributes of the Software Requirements KA

The life cycle of XP defines a clear process for collecting, specifying, analysing and validating requirements through a software project. In this process, actors and used artifacts are clearly defined and explained, therefore allowing one to conclude that XP **adequately satisfies** the sub-attribute *2. Requirements Process*.

The primary technique used for requirements elicitation in XP is the creation of user stories by the client, which is seen as a team member of the project. The client works together with the other team members to create user stories, to define functional and acceptance tests, and to prioritise requirements. In XP, requirements are not a monolithic document, but rather include a collection of user stories and a set of functional, acceptance and unit tests, incrementally defined [14]. XP **adequately satisfies** the sub-attribute *3. Requirements Elicitation*.

XP does not present any practice or technique that addresses the detection and negotiation of conflicts. This aspect of XP hinders its adoption when the client is not aware of the requirements or when the requirements, specified by several stakeholders, are in conflict. Regarding the classification of requirements, XP assigns a priority to a given requirement based only on its business value for the client. Since XP does not propose techniques to prevent and solve conflicts among the requirements and only provides a simple method for classifying and prioritising requirements, XP **partially satisfies** the sub-attribute *4. Requirements Analysis*.

Requirements validation in XP is performed by the user defining a set of functional and acceptance tests. Thus, XP **adequately satisfies** the sub-attribute *5. Requirements Validation*. Table 6 presents a summary of the classification of XP under the attribute Software Requirements.

4.1.2. Scrum. Several basic concepts of software requirements are present in Scrum, which defines the phases of the life cycle and a set of practices focused in requirements elicitation. Additionally, elements integrating the Product Backlog may represent both system or product functionalities to be implemented, functional or non-functional requirements, research activities, or errors and defects to be corrected. Scrum quantifies and qualifies requirements, presenting a clear definition of what each different type of requirement is, therefore **adequately satisfying** the sub-attribute *1. Software Requirements Fundamentals*.

Sub-attribute	*Classif.*	*Aspects of XP that satisfy the sub-attribute*
Software Requirements Fundamentals	**PS**	• Definition of the concepts related to software requirements; • No distinction between different types of requirements.
Requirements Process	**AS**	• Definition of a process to collect, specify, analyze and validate requirements, explicitly defining the actors and activities to be undertaken.
Requirements Elicitation	**AS**	• On-site client; • User Stories.
Requirements Analysis	**PS**	• Classification of requirements by setting priorities (business value criteria); • No techniques for detection and resolution of conflicts between requirements.
Requirements Validation	**AS**	• Functional and acceptance tests written by the client.

Table 6: Analysis performed under the attribute Software Requirements for XP

The requirements process in Scrum, starts with the pre-game phase where the Product Backlog is created. During this phase, the client works together with the product owner to associate a priority level to each element of the backlog. At the start of each sprint, a sprint planning meeting takes place, where a subset of the elements of the Product Backlog is selected to be implemented during that sprint. Each of these elements is divided by the Scrum team into activities to be performed, which are integrated in the sprint backlog. This short description of the requirements process associated with Scrum shows that it **adequately satisfies** the sub-attribute *2. Requirements Process*.

Scrum presents two main methods for requirements elicitation, therefore **adequately satisfying** the sub-attribute *3. Requirements Elicitation*: (1) close interaction between the client and the project team during the pre-game phase, and (2) the possibility for any of the involved parties of the project to add a new item to the Product Backlog.

Regarding requirements classification, the analysed methods are very similar. Both XP and Scrum associate a given priority to each of the elicited requirements based on the business value that each requirement has for the client, which may in some cases reveal to be a simplistic prioritization system. Additionally, Scrum does not suggest techniques for detection and resolution of conflicts among software requirements, therefore **partially satisfying** the sub-attribute *4. Requirements Analysis*.

Although the requirements process of Scrum tackles requirements validation, no practice or technique for validating elicited requirements is presented, therefore the sub-attribute *5. Requirements Validation* is **not satisfied** by Scrum. A summary of the classification of Scrum under the attribute Software Requirements is presented in table 7.

4.2. Software Construction

The Software Construction KA refers to the activities related to the creation of software through a combination of coding, verification, unit testing, integration testing, and debugging [13]. This KA comprises 3 sub-areas. For this technique, we have chosen the sub-area Software Construction Fundamentals. This sub-area encompasses 4 sub-attributes focused on software design and implementation: *1. Minimizing Complexity*, *2. Anticipating Change*, *3. Constructing for Verification* and *4. Standards in Construction*. For this study, the sub-attribute *2. Anticipating Change* was replaced for *2. Response to Changes*. Agile methods characterize themselves for embracing/responding to unexpected changes, rather that try to anticipate those same changes. Although clearly different, replacing the sub-attribute *2. Anticipating Change* for *2. Response to Changes* would better address the agile method characteristics and focus. An analysis under this perspective aims to identify how XP and Scrum address the implementation of a software project, with a focus on the sub-attributes in table 8.

4.2.1. XP.

XP proposes two practices that address the sub-attribute *1. Minimizing Complexity*: simple design and refactoring. Simple design practice is intended to prevent the introduction of unnecessary complexity in the implemented code, thereby facilitating the comprehension and the maintenance of the code. The practice of refactoring allows (1) to create a more readable and easy to understand code by removing unnecessary complexity, (2) to eliminate redundancy and duplicated functionality and (3) to increase the modularity and scalability, thus promoting code reuse. Additionally, one of the values of XP is simplicity, as opposed to complexity. Thus, XP **adequately satisfies** the sub-attribute *1. Minimizing Complexity*.

The sub-attribute *2. Response to Changes* reflects the agility and rapid response to changes in a software project, two of the main aspects of agile methods. Through a set of practices, such as collective code ownership, refactoring and continuous integration, XP improves its ability to respond to changes, therefore **adequately satisfying** the sub-attribute *2. Response to Changes*. As stated before, collective ownership avoids unnecessary delays when writing/changing code. The practice of refactoring indirectly affects the responsiveness to changes in the course of a software project. Refactoring aims to create simple and readable code, and the task of making changes to the implemented code is greatly facilitated when the code is readable and simple. The practice of continuous integration supports the ability of responding to changes, ensuring product integrity and that changes to existing code and the implementation of new features does not compromise the work already done.

The existence of practices such as test-driven development and pair programming, in XP, creates a solid infrastructure that consists in a set of tests whose implementation accompanies or precedes the actual implementation of the code. This infrastructure supports the validation and verification of the created code. The combination of these two practices allows the rapid detection of bugs or errors during the writing of the code and the execution of the tests, therefore **adequately satisfying** the sub-attribute *3. Constructing for Verification*.

Sub-attribute	*Classific*	*Aspects of Scrum that satisfy the sub-attribute*
Software Requirements Fundamentals	AS	• Definition of software requirements related concepts; • Distinction among different types of requirements.
Requirements Process	AS	• Pre-game phase; • Sprint planning meeting; • Definition of a process to collect, specify, analyse and validate requirements, explicitly identifying the actors and the activities to be undertaken.
Requirements Elicitation	AS	• Close interaction between the client and project team in early stages; • Product backlog; • Sprint backlog; • Possibility for any of the evolved entities to add a new element to the Product Backlog.
Requirements Analysis	PS	• Classification of requirements by setting priorities (business value criteria); • No techniques for detection and resolution of conflicts between requirements.
Requirements Validation	NS	-

Table 7: Analysis performed under the attribute Software Requirements for Scrum

Sub-attribute	*Description*
1. Minimizing Complexity	Refers to the use of techniques and practices to minimize the complexity of the implemented code
2. Response to Changes	Refers to the use of techniques and practices that aim at improving the ability to respond to possible changes
3. Constructing for Verification	Refers to the adoption of techniques and practices in code implementation, which allows a rapid detection of bugs and/or errors during the writing of the code and the execution of the tests
4. Standards in Construction	Refers to the use of standards on a software project

Table 8: Sub-attributes of the Software Construction KA

The use of coding standards, practice proposed by XP, aims to ease the interpretation of the code by programmers other than the its authors. Additionally, applying other practices of XP, such as refactoring and pair programming, would be considerably more difficult without the use of coding standards. The use of standards on a software project should not be restricted to the implementation phase, but can also include the other phases, like requirements, design, test, or maintenance. Since the use of standards is only explicitly proposed for the implementation phase, XP **partially satisfies** the sub-attribute *4. Standards in Construction*. A summary of the XP analysis performed under the attribute Software Construction is presented in table 9.

4.2.2. Scrum. A detailed analysis to the practices proposed by Scrum reveals that this agile method focuses mainly on the management activities of the software projects, and can also be applied to other project types. Scrum appears with the objective of improving the software engineering concepts used in project management, identify problems and restrictions in the development process and in the implementation practices used. Thus, the implementation and construction process itself is not addressed by Scrum, leaving to the cross-functional teams to choose the practices used in architectural design, implementation, product configuration, testing and integration. Thus, all sub-attributes of the KA *Software Construction* are **not satisfied** by Scrum. A summary of the analysis on Scrum performed under the attribute Software Construction is presented in table 10.

Sub-attribute	*Classification*	*Aspects of XP that satisfy the sub-attribute*
Minimizing Complexity	AS	• XP values; • Simple Design; • Refactoring.
Response to Changes	AS	• Collective Code Ownership; • Refactoring; • Continuous Integration.
Constructing for Verification	AS	• Test-Driven Development; • Pair Programming.
Standards in Construction	PS	• Coding Standards.

Table 9: Summary of the analysis on XP for attribute Software Construction

Sub-attribute	*Classification*	*Aspects of Scrum that satisfy the sub-attribute*
Minimizing Complexity	NS	-
Response to Changes	NS	-
Constructing for Verification	NS	-
Standards in Construction	NS	-

Table 10: Summary of the analysis on Scrum for attribute Software Construction

4.3. Synthesis of the Results

Under the attribute *Software Requirements*, figure 1(a) shows that 3 out of 5 of the selected sub-attributes are **adequately satisfied** by XP and that 2 of those attributes are **partially satisfied**. Figure 1(b) illustrates the fact that 3 of those attributes are **adequately satisfied**, 1 is **partially satisfied**, and the last one is **not satisfied** by Scrum. The results of XP derive mainly from the close contact between the technical team and the client. The importance given to this relationship is reflected in the fact that XP address all sub-attributes of the *Software Requirements* KA.

Regarding the attribute *Software Construction*, the analysis concludes that XP **adequately satisfies** 3 out of the 4 selected sub-attributes and **partially satisfies** 1 of those same sub-attributes, while Scrum does **not satisfy** any of the selected sub-attributes. The fact that Scrum does not explicitly suggest any practice oriented towards implementation, delegating the choice of which practices to apply to the cross-functional teams, shows a clear difference between XP and Scrum under the attribute *Software Construction*.

Attributes	**XP**			**Scrum**		
	AS	*PS*	*NS*	*AS*	*PS*	*NS*
Software Requirements	60%	40%	0%	60%	20%	20%
Construction of Software	75%	25%	0%	0%	0%	100%
Software Testing	100%	0%	0%	0%	0%	100%
Software Engineering Management	80%	0%	20%	80%	20%	0%
Agile Principles - Proposed Practices relation	75%	8%	17%	50%	17%	33%

Table 11: Complete results when comparing XP and Scrum under five attributes

XP puts a great emphasis on testing. Based on *test-driven development*, XP places tests as the foundations of software development. The results of the analysis under the attribute *Software Testing* reflect this aspect of XP and the importance it places on creating and running tests in a software project. XP analysis under the attribute *Software Testing* revels that

XP **adequately satisfies** all sub-attributes of this KA. Scrum does **not satisfy** any of the sub-attributes of *Software Testing*.

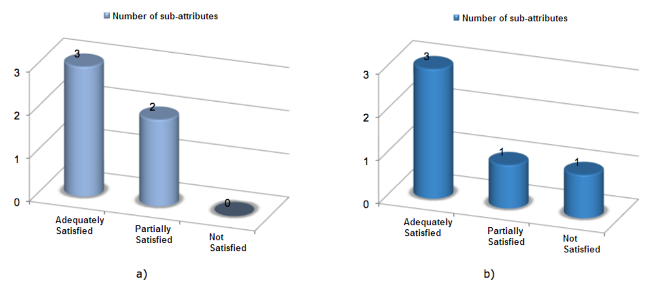

Figure 1: Results of the analysis to (a) XP and (b) Scrum, for attribute Software Requirements

Under the attribute *Software Engineering Management*, XP **adequately satisfies** 4 of the 5 selected sub-attributes and only the sub-attribute Review and Evaluation was **not satisfied**. When compared to the same set of sub-attributes Scrum reveals itself more complete than XP, **adequately satisfying** 4 of the selected sub-attributes and **partially satisfying** one of those same sub-attributes. Furthermore, the analysis shows that Scrum tackles this part of a software project in a much more detailed and precise way when compared to XP.

Under the attribute *Agile Principles vs. Proposed Practices relation*, results show that XP **adequately satisfies** 7 of the principles presented in the Agile Manifesto, while Scrum only **adequately satisfies** 5 of those same attributes. Considered the first agile method and with its origin previous to the creation of the Agile Manifesto, XP has clearly influenced a considerable number of the principles that constitute the manifesto and the concepts which characterise agile methods. Therefore, it is not a surprise that, under this attribute, XP adequately satisfies a greater number of principles of the Agile Manifesto when compared with Scrum. Table 11 synthesises all the obtained results.

5. Conclusions and Future Work

The increasing challenges that need to be addressed in software development projects can only be overcome by professionals with experience and excellent skills, both hard and soft. There are no two equal projects and therefore the approaches and the practices that have proven effective previously may not be in the future. Thus, it is essential to gain a thorough knowledge of the existing development methods when deciding which one(s) to employ, based on the characteristics of the projects and on the competences of the team members. Thus, one important issue is to decide on the development method (or the combination of practices advocated by various development methods) that best suits the software project under consideration in a given situation.

This manuscript contributes to this challenge by presenting a technique and the corresponding tool that permits one to classify and compare software development methods under a set of relevant features and attributes. This set includes attributes related to four SWEBOK KAs and to the agile principles defined in the Agile Manifesto. With these attributes, by analysing the practices proposed by each method, we are able to assess a given agile method in relation to (1) the coverage degree for the considered KAs and (2) the agility degree. Since the technique was especially conceived to compare agile methods, we exemplify its usage in a comparison between XP and Scrum.

As future work, the following actions are planned:

- To apply Pareto analysis to compare agile methods; one method could cover less a set of attributes, but cover more another one. Thus, based on the context, the developer chooses one or the other.
- To extend the set of analysed methods to include other agile methods, such as Adaptive Software Development (ASD), Agile Modeling (AM), Dynamic Systems Development Method (DSDM) and Feature-Driven Development (FDD);
- To include in the analysis the existence of tools to support the various agile methods;
- To extend the set of SWEBOK KAs used to classify and compare the agile methods.

References

[1] A. Qumer, B. Henderson-Sellers. A framework to support the evaluation, adoption and improvement of agile methods in practice. *J. Syst. Softw.*, 81(11):1899–1919, 2008.

[2] A. Qumer, B. Henderson-Sellers. An evaluation of the degree of agility in six agile methods and its applicability for method engineering. *Inform. Softw. Tech.*, 50(4):280–295, 2008.

[3] J. McAvoy, D. Sammon, I. Owens. A Simple Tool to Assist in Agile Methodology Adoption Decisions. *J. Decis. Syst.*, 16(4):451–468, 2007.

[4] V. R. Basili, A. J. Turner. Iterative Enhancement: A Practical Technique for Software Development. *IEEE Trans. on Softw. Eng.*, 1(4):390–396, 1975.

[5] M. C. Paulk. Extreme Programming from a CMM Perspective. *IEEE Softw.*, 18(6):19–26, 2001.

[6] K. Beck. Embracing Change with Extreme Programming. *Computer*, 32(10):70–77, 1999.

[7] K. Beck, C. Andres. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2nd edition, 2004.

[8] H. Takeuchi, I. Nonaka. The New New Product Development Game. *Harvard Bus. Rev.*, 64(1):137–146, 1986.

[9] K. Schwaber. Scrum Development Process. *OOPSLA'95 Business Object Design and Implementation Workshop*. Springer, 1995.

[10] B. Boehm, R. Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley, 2003.

[11] H. G. Sol. A feature analysis of information systems design methodologies: methodological considerations. *Working Conf. on Feature Analysis of Information Systems Design Methodologies*, pp. 1–8, 1983.

[12] P. Abrahamsson, O. Salo, J. Ronkainen, J. Warsta. Agile Software Development Methods: Review and Analysis. Technical Report 478, VTT Centre of Finland, 2002.

[13] A. Abran, P. Bourque, R. Dupuis, J. W. Moore (eds). *Guide to the Software Engineering Body of Knowledge - SWEBOK*. IEEE Press, 2004.

[14] R. Duncan. The Quality of Requirements in Extreme Programming. *J. Def. Softw. Eng.*, 2001.