CEFAM: Comprehensive Evaluation Framework for Agile Methodologies

Masoumeh Taromirad, Raman Ramsin
Department of Computer Engineering
Sharif University of Technology
taromi@ce.sharif.edu, ramsin@sharif.edu

Abstract

Agile Software Development is regarded as an effective and efficient approach, mainly due to its ability to accommodate rapidly changing requirements, and to cope with modern software development challenges. There is therefore a strong tendency to use agile software development methodologies where applicable; however, the sheer number of existing agile methodologies and their variants hinders the selection of an appropriate agile methodology or method chunk. Methodology evaluation tools address this problem through providing detailed evaluations, yet no comprehensive evaluation framework is available for agile methodologies. We introduce the Comprehensive Evaluation Framework for Agile Methodologies (CEFAM) as an evaluation tool for project managers and method engineers. The hierarchical (and mostly quantitative) evaluation criterion set introduced in this evaluation framework enhances the usability of the framework and provides results that are precise enough to be useful for the selection, adaptation and construction of agile methodologies.

1. Introduction

Agile software development methodologies claim to speed up the delivery of software solutions to a client's rapidly changing requirements. Project managers are therefore expected to prefer agile software development methodologies over their plan-driven counterparts. However, in reality, agile software development methodologies are not used as commonly as expected, and are often applied to the wrong context [2, 3]. One of the reasons behind this may be the lack of appropriate technical and management tools.

Project managers need to select the most appropriate agile methodology for their projects. Method engineers, on the other hand, need to construct a tailored-to-fit agile methodology. In both situations, appropriate tools are indispensable. Such tools should consider existing challenges and project-specific parameters in order to help project managers select a suitable agile methodology, and assist method engineers in choosing method-fragments to ultimately assemble a bespoke agile methodology. In order to achieve this, a tool has to identify the weaknesses, capabilities, similarities, and differences of agile software development methodologies; evaluation frameworks and methods are integral parts of such tools.

Previous studies have shown that existing evaluation frameworks do not satisfy existing requirements and challenges [1]. Hence, the aim of this paper is to introduce a comprehensive evaluation framework for agile methodologies that addresses the requirements of such frameworks. The *Comprehensive Evaluation Framework for Agile Methodologies (CEFAM)* proposed herein strives to provide full coverage of the aspects and characteristics regarded as important in an agile software development methodology.

The rest of this paper is structured as follows: Section 2 presents a short overview of the background and related work; in Section 3, existing problems are explained in detail, and CEFAM and the definition method are introduced; Section 4 contains the proposed evaluation criterion set, and Section 5 shows the results of evaluating the eXtreme Programming (XP) methodology using CEFAM; the final section summarizes the key achievements of this study and suggests ways for furthering this research.

2. Background and related work

Several evaluation frameworks/methods exist for agile methodologies, each of which has focused on specific characteristics or limited aspects (views) of these methodologies. We provide a brief overview of these frameworks in this section. The interested reader is referred to [1] for an in-depth analysis.

Abrahamsson et al. have introduced a structure in which the process, roles and responsibilities, practices, status of adoption, experiences, scope of use, and current research regarding each method are identified

1550-6215/09 $25.00 © 2009 IEEE
DOI 10.1109/SEW.2008.19

[4]. In another paper, Abrahamsson et al. have proposed an analytical framework for the analysis of existing agile methods [5], using software development notions – such as lifecycle coverage (including the process), project management, abstract principles vs. concrete guidance, universally predefined vs. situation appropriate, and empirical evidence – as "analytical lenses".

With a focus on the critical role of Software Configuration Management (SCM) in software development, especially in agile methods, Koskela has introduced a number of specialized "analytical lenses" – including SCM approach, SCM planning, configuration identification, change management, and SCM tools – to analyze the state of software configuration management in agile methodologies [6].

Williams et al. have provided a benchmark for assessing XP practices adopted in organizations [7]. This evaluation framework (called XP-EF) is composed of three parts: XP Context Factors (XP-cf) to record essential context information about a project, XP Adherence Metrics (XP-am) to concretely and comparatively express the practices a team utilizes, and XP Outcome Measures (XP-om) to assess and report a team's outcome when using a full or partial set of XP practices.

Germain et al. have provided an empirical comparison between an engineering-based process (Unified Process for Education – UPEDU) and an agile process built around XP principles, mainly through comparing and analyzing the work and time spent in each of their "cognitive" activities [8].

One of the latest works in this context is 4-DAT, a framework-based assessment tool for the analysis and comparison of agile methods [9]. 4-DAT provides evaluation criteria for the detailed assessment of agile software development methods through defining four dimensions: 1) Method scope characterization; 2) Agility characterization; 3) Agile Values characterization; and 4) Software Process characterization.

In addition to these evaluation frameworks, there exist certain characteristics that are inherently associated with agile methodologies, and which can be used as evaluation criteria [2, 3, 10, 11]: the Agile Manifesto [12] and Agile Principles [13] delineate the fundamental characteristics of agile methods; Conboy et al. have introduced *flexibility* and *leanness* as the essential properties of any agile software development method [14]; and Boehm and Turner have introduced *project size*, *project criticality*, *dynamism*, *personnel*, and *culture* as crucial variables in agile methods [15, 16].

Unlike agile methodologies, there are many relatively comprehensive and mature evaluation frameworks aimed at other types of software development methodologies; examples include object-oriented- [17, 18, 19] and agent-oriented methodologies [20, 21, 22]. As agile methodologies gain widespread popularity, it is becoming increasingly important that an adequate evaluation framework be developed for agile methodologies.

We have previously introduced a set of meta-criteria that describe the features and characteristics which an appropriate evaluation criterion set should satisfy in order to provide valuable results when applied to agile software development methodologies. The meta-criteria were applied to several existing evaluation frameworks, showing that they are lacking in several aspects, especially as to comprehensiveness [1]. One important deficiency unearthed was that most of the frameworks evaluated do not adequately address agility issues in their criteria. Furthermore, despite the importance of the usage context – as it addresses the main concerns of project managers – it has been neglected or only partially addressed in most evaluation frameworks.

Adequate coverage of quantitative metrics is yet another important feature neglected in existing evaluation frameworks. Most of the quantitative metrics introduced in the context of agile methodologies provide very limited coverage of the crucial aspects. Most metrics deal with running or finished processes (methodology instances), evaluating the performance of a specific process or technique, and assessing the complexity and stability of the models produced [5, 23, 24, 25, 26, 27]. There are few metrics available for evaluating a methodology independent of its instances, and these mainly focus on modeling some part of the methodology with a specific language, and then measuring the presence of certain characteristics (such as complexity) in the resulting model [7, 28, 29, 30, 31].

3. Proposed evaluation framework: CEFAM

In this section, we introduce our proposed Comprehensive Evaluation Framework for Agile Methodologies (CEFAM). CEFAM addresses the shortcomings commonly encountered in existing frameworks and strives to comply with the meta-criteria defined in [1]. The following are the primary objectives of CEFAM:

1. Supporting evaluation in such a way as to accentuate the similarities, differences,

features, and applications of agile software development methodologies.

2. Supporting the evaluation of agile methodologies so as to facilitate the construction of custom methodologies.

Accordingly, a multi-step method was chosen to define the main part of the framework, i.e., the evaluation criterion set. Criteria were first defined targeting each of the main aspects of agile software development methodologies, as defined by the meta-criteria of [1], to ensure the comprehensiveness of the evaluation framework produced. As these criteria were defined independently, the resulting set was not a proper evaluation criterion set and had to be refined. Improvements were made through removing inconsistencies, conflicts, overlappings, and redundancies, adding extra cross-aspect criteria (criteria that cover more than one aspect), and restructuring the resulting set. The criterion set produced was comprehensive according to the meta-criteria, but to make sure of its completeness and precision, it was further refined through iterative application to a prominent agile methodology: eXtreme Programming (XP).

4. CEFAM evaluation criteria

Evaluation criteria play an essential role in any evaluation and comprise the most important part of evaluation frameworks. The proposed set of evaluation criteria is introduced in this section.

The evaluation criteria have been divided into five groups. Four of the divisions group the criteria according to the context that they target: *Process*, *Modeling Language*, *Agility*, and *Usage*; the fifth group includes *Cross-Context* criteria: criteria which cover the overlappings among the other four groups, typically transcending context-related issues. Moreover, each group has been further divided into subgroups, each of which contains evaluation criteria corresponding to a specific view of the relevant context. For example, there are certain criteria in the Process group which evaluate the process part of the methodology with respect to its *definition*; these have therefore been grouped together as a subgroup of the Process group. Figure 1, shows the hierarchical structure of the evaluation framework. This hierarchy can help the evaluators in selecting criteria that better suit their evaluation goals.

In order to provide valuable and comparable evaluation results, every effort has been made to define evaluation criteria as quantitative metrics where possible. For those criteria which are not of a quantitative nature, discrete values have been defined for the evaluation results so that measurability is maintained. In addition, to enhance the understandability of the results of applying quantitative criteria, descriptive levels (typically: *Unacceptable*, *Low*, *Medium*, and *High*) have also been used to categorize the results. In most of the quantitative criteria proposed, the evaluation result is a real number greater than zero and less that 1.0; in such cases, descriptive levels have been defined as follows: Unacceptable ≤ 0.25; $0.25 <$ Low ≤ 0.5; $0.5 <$ Medium ≤ 0.75; $0.75 <$ High ≤ 1.0. Note that to calculate compound evaluation results over quantitative results, quantitative values will be used instead of their descriptive equivalents.

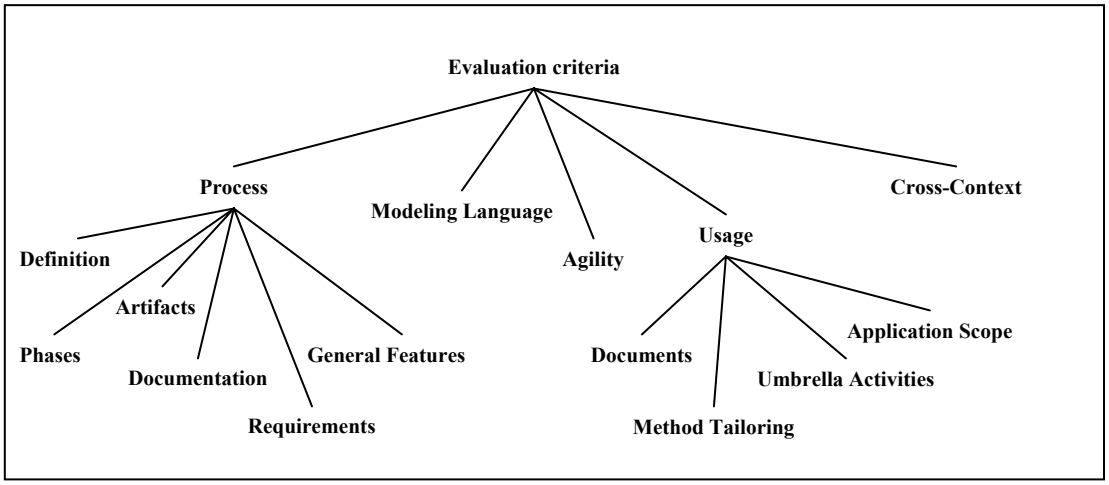

Figure 1. CEFAM hierarchy.

4.1. Process

Process evaluation criteria focus on the process part of the methodology. This group is divided into six subcategories: Definition, Phases, Artifacts, Requirements, Documentation, and General Features. The evaluation criteria have been introduced in Table 1, with definitions and domain values provided in separate columns. The table also contains the results of applying the criteria to the XP methodology.

Process evaluation criteria are divided into six subgroups: *Definition* criteria focus on the definition characteristics of the development process; *Phases* and *Artifacts* criteria consider development lifecycle and activities and products respectively; *Requirements* criteria analyze requirements engineering issues in the development process; *Documentation* criteria focus on the documents existing on the development process; and criteria in the *General Features* category focus on the development process in its entirety.

Table 1. Process evaluation criteria.

Criteria	Description	Domain Values	XP Evaluation
Definition			
Explicitness and Unambiguity	Is the development process defined explicitly and unambiguously?	Yes, No	No (the introduced process is just a typical example)
Rationale	Has the process been rationalized through providing extensive and precise explanations? [5]	Yes, No (why)	No (XP has neglected detailing the process in order to remain abstract, thereby damaging rationalization)
Completeness	A complete process definition includes definitions for: *development lifecycle*, *roles*, *activities*, *modeling language*, *artifacts*, *practices/techniques*, *rules*, and *umbrella activities*.	Ratio of the number of existing definitions to the total.	(1+1+1+0+1+1+1+0)/8 6/8 (High)
Phases			
Generic development lifecycle coverage	Which phases of the generic development lifecycle are covered by the development process? Generic phases include: *Inception*, *Analysis*, *Design*, *Implementation*, *Test*, *Deployment*, *Maintenance*, *Support*, and *Postmortem*.	Ratio of the number of covered phases to the total number of generic phases.	(0+1+0+1+1+1+1+0+1)/9 6/9 (Medium)
Smooth transition	Is the transition between phases smooth? What techniques are prescribed for providing smoothness of transition?	Yes (techniques), No (counterexamples)	Yes (through short iterations)
Seamless transition	Are there any gaps between phases? What techniques are prescribed for enhancing seamlessness?	Yes (techniques), No (counterexamples)	No (there is a gap between analysis and development)
Development style	What is the development style?	Iterative, Incremental, Rapid, etc.	Iterative, Rapid
Artifacts			
Adequate products	Does the development process produce the products typically associated with the generic development activities (*Feasibility analysis, Requirement specification, Design, Modeling, Documentation, Test, Training,* and *Deployment*)?	Ratio of product types supported to the ideal number of product types.	(0+1+0+1+1+1+0+0)/8 4/8 (Low)
Modeling coverage	Do the products include models (analysis and design)?	Yes (models), No	No
Consistency	Do the products complement each other?	High, Medium (overlappings exist that can result in inconsistencies), Low	Medium
Tangibility/Visibility/Testability	Are the products tangible, understandable, and testable to end users?	High, Medium, Low	Medium
Supported views	Which generic views do the products support?	Structural, Behavioral, Functional	Functional (through user stories)
Abstraction levels	Which abstraction levels are provided by the products?	System/Subsystem/Package/ Intra-object/Inter-object, Logical/Physical, Task/Process, Problem/Solution/ Implementation	Problem/Solution/ Implementation
Standards	Are there any specific standards for the products?	Yes (standards), No	Yes (coding standard)
Requirements			
Requirements elicitation	How are the requirements collected?	Related activities, roles, artifacts	Through user stories, written by customers in the first phase and revised at the start of each iteration.
Requirements specification format	How are the requirements specified?	User story, Feature, Use-case, Usage scenario	User story
Process based on functional/non-functional requirements	Is the development process based on the requirements?	Yes (techniques), No	Yes (development process centered around the requirements specification)
Non-functional requirements verification	How are the non-functional requirements addressed?	Techniques	Non-functional requirements are captured in user stories
Traceability	Can the products be traced to the requirements?	Yes (techniques), No	Yes (through user stories)
Requirements change	Does the development process let changes in requirements?	Yes (techniques), No	Yes (user stories are updated at the beginning of any iteration)
Requirements prioritization	On what basis are the requirements prioritized?	Architectural value, Functional value, Business value, Development risk	Business value

Table 1. Process evaluation criteria (contd.).

Criteria	Description	Domain Values	XP Evaluation
Documents			
Available and published documents	Is the development process published and available to users?	Published and available, published but not available, not published and not available.	Published and available
Process enactment documentation	Is the running process documented?	Yes, No	Yes
General Features			
Size/Complexity	Size/Complexity is defined as a function of building blocks of the development process.	The function is defined by the evaluators according to their preferences. We have defined it as the total number of practices, roles, products, and phases/stages.	Phases/stages: 6 Roles: 7 Products: 5 Practices: 12 Total: 30
Completeness	Completeness is defined as a function of process definition completeness, coverage of generic lifecycle, and adequate products.	The function is defined by the evaluators according to their preferences. We have defined it as the weighted sum of the relevant criteria (equal weights have been assigned in our evaluation of XP).	1/4 (6/8 + 6/9 + 3/8) 0.45 (Low)
Practicality	Is the process practical? (Based on issues affecting practicality, such as support for umbrella activities, independence from specific tools, pragmatic techniques, concrete rules, and non-overlapping activities)	High, Medium, Low	Medium (E.g. collective code ownership, standup meetings)
Practicability	Is the process development practicable? (Through providing effective activities or techniques such as suitability filters and instantiation/adaptation methods.)	High, Medium, Low	Low

4.2. Modeling language

Generally, agile software development methodologies pay little attention to modeling and modeling languages, in some cases leaving it out altogether. Nevertheless, the modeling language should be considered in any usable (practical) methodology, even if partially or indirectly. Therefore, we have defined criteria for evaluating the modeling language part of agile methodologies. There are no subgroups in this group. As shown in Table 2, modeling language evaluation criteria have been defined qualitatively and at a high level. The table also contains the results of applying the criteria to the XP methodology, which prescribes nothing as to modeling and modeling language.

Table 2. Modeling language evaluation criteria.

Criteria	Description	Domain Values	XP Evaluation
Simple to learn and use	Is the modeling language simple to learn and use?	Yes, No	-
Power of language	Is the modeling language powerful? E.g., Support of semantics, multiple views, and model execution.	Yes (examples), No	-
Handling model inconsistencies	Does the language provide techniques for handling inconsistencies?	Yes (techniques), No	-
Managing model complexities	Does the language provide methods for managing complexities?	Yes (methods), No	-

4.3. Agility

This category of evaluation criteria evaluates the characteristics attributed to and contributing to a methodology's agility. These criteria have been defined based on the Agile Manifesto, Agile Principles, and papers presenting common agile traits. The criteria can be used to evaluate the degree of agility in any software

Table 3. Agility evaluation criteria.

Criteria	Description	Domain Values	XP Evaluation
Speed	How quickly does the methodology produce results?	1/(iteration length (in days) + deployment interval (in days))	1/(14 + 0) = 1/14 (iteration between 1 to 3 weeks)
Sustainability	Are speed and quality maintained until the end? Are they controlled or monitored?	Yes (techniques), No	Yes (controlling project velocity, TDD - test-driven development, refactoring)
Flexibility	Are expected/unexpected changes captured and handled in the project?	Ratio of the number of supporting activities and practices to the total	15/18 (High)
Learning	Does the process "learn" from past projects and previous iterations?	Ratio of the number of supporting activities and practices to the total	16/18 (High)
Responsiveness	Does the method provide feedback?	Ratio of the number of supporting activities and practices to the total	15/18 (High)
Leanness	Does the method value shorter time spans, using economical and simple quality-assured means for production?	Ratio of the number of supporting activities and practices to the total	6/18 (Low)
Lightness and simplicity	How light and simple is the development process?	1 / process complexity	1/30 = 0.03
Technical quality	How is technical quality monitored and controlled during the development?	Techniques and metrics	Yes (Coding standard, TDD, Pair programming)
Active user collaboration	How involved are the customers in the development process?	Related role(s) and responsibilities	On-site customer

development methodology, even non-agile ones.

The evaluation criteria are shown in Table 3. The table also contains the results of applying the criteria to the XP methodology.

4.4. Usage

The usage view of a methodology addresses the practical aspects of a methodology. This view is complementary to the definition of a methodology (both process and modeling language). A methodology which does not consider practical issues is at best difficult to use in practice. Support for umbrella activities, adequate application scope, scalability, and flexibility are examples of important usage issues which a project manager encounters in the real world.

Usage evaluation criteria are divided into four subgroups: *Application Scope* criteria address project-specific parameters mostly useful to project managers when selecting a methodology for a specific project; *Umbrella Activities* criteria focus on the activities required for enacting a methodology in the real world; *Method Tailoring* criteria address method customization issues in a methodology; and *Documents* criteria analyze the methodology as to the existence of usage guides, empirical evidence, and experience reports.

Table 4 introduces and describes these criteria. As inferred from the table, evaluation can be done through careful study of a methodology's definition. The table also contains the results of applying the criteria to the XP methodology.

Table 4. Usage evaluation criteria.

Criteria	Sub-criteria/Description	Domain Values	XP Evaluation
Application Scope			
Project	Software type	Customizable, Specific, Universal	-
	Size	Small, Medium, Large	Small, Medium
	Length	Month	-
	Domain	System, Real-time, Business, Engineering and scientific, Embedded, Personal computer, Web-based, Artificial intelligence [32]	-
	Culture	Percentage of thriving on chaos versus order [15]	-
	Dynamism	Percentage of requirements change/month [15]	-
	Complexity (computational complexity)	High (scientific and complex), Medium (Business-oriented and IS), Low (simple and personal usage)	Medium
	Criticality (loss due to impact of defects)	Comfort, Discretionary funds, Essential funds, Life	-
	Priority (main goal of the project) [33]	Productivity, Visibility, Repeatability, Correctness, Liability	Productivity
	Constraints	Specific constraints on the project.	-
Development team	Size	Number of personnel per team	<10
	Education level	-1, 1B, 1A, 2, 3 [16]	>1A
	Experience (In software development generally)	High, Medium, Low (Based on years; typically, it can be divided into three levels: 2-4, 5-7, 8-X)	High
	Skill in domain	High, Medium, Low (Based on years; typically, it can be divided into three levels: 2-4, 5-7, 8-X)	High
	Skill in development language	High, Medium, Low (Based on years; typically, it can be divided into three levels: 2-4, 5-7, 8-X)	High
Ergonomic	Physical layout	Distributed, Collocated	Collocated
Geographical	Number and location of development teams and customers	Distributed, Remote, Local, Multinational, different time zone; Single, Multiple	Distributed and co-located single or multiple teams
Technical	Programming language		OO programming languages
	Programming style	Simple, Complex	Simple
	Abstraction techniques	Object-oriented, Agent-oriented, etc.	Object-oriented
	Obligatory development tools		Collective code ownership
	Test and debug methods		Automated black-box acceptance tests
Managerial	Management team size	Small, Medium, Large	Medium
	Management experience	High, Medium, Low	High
	Team management approach	Centralized, Distributed	Centralized
	Resource allocation method		Planning Game
	Project culture		-
	Business culture	Collaborative, Cooperative, Non-collaborative	Collaborative, Cooperative
	Team values (preferable interaction method)		Informal (daily standup meetings)
	Customer collaboration method		On-site customer
	Quantitative metrics		-

Table 4. Usage evaluation criteria (contd.).

Criteria	Sub-criteria/Description	Domain Values	XP Evaluation
Umbrella Activities			
Project management	Support for development process management; including planning, scheduling, controlling and monitoring, and process review.	Ratio of the number of covered activities to the total.	(1+1+1+0)/4 3/4 (Medium)
Software configuration management	Support for configuration management approaches and tools.	SCM approach, SCM planning, Configuration identification, Change management, SCM tools [6]	Partially considered (Collective code ownership, small releases, and continuous integration)
Team management	Does the methodology provide any process for team and people management?	Yes (techniques), No	Yes (Pair programming, cyclic team assignment)
Quality assurance	Support for quality assurance techniques such as technical review, continuous verification and validation, and strategies/techniques enhancing requirements traceability.	Yes (techniques), No	Yes (Pair programming, Continuous integration, Refactoring, Test-driven development, Coding standards, On-site customer)
Risk management	Support for risk management techniques such as feasibility analysis, risk-based planning, active user involvement, continuous verification and validation, iterative process/product/plan reviews, and continuous integration.	Yes (techniques), No	Yes (Planning game, Small and short release, Metaphor, Testing, Continues integration, On-site customer)
Method Tailoring			
Adaptation and customization	Does the methodology provide methods for customizing it based on the parameters of the project at hand?	Yes (method), No	No
Flexibility	Does the methodology allow the process and modeling language to be changed during its execution?	Yes (how), No	No
Scalability	Is the methodology suitable for projects with different sizes, criticalities, and complexities?	Yes, No	No (lack of evidence)
Extensibility	Does the methodology provide any extension points?	Yes (what), No	No
Integration with other methodologies	If the methodology does not completely cover the generic development lifecycle, it should provide some method to integrate it with other methodologies.	Not needed, Needed but not provided, Needed and provided	Not needed
Documents			
Tutorials and training documents	Are tutorials and training documents available?	Yes, No	Yes
Empirical evidence	Does empirical evidence exist?	Yes, No	Yes

4.5. Cross-Context

The evaluation criteria belonging to this group address issues that are associated with more than one context (process, modeling language, agility and usage) at the same time, or focus on the methodology as a whole. For instance, the *Status* criterion addresses the methodology as a whole. These criteria have been shown in Table 5. The table also contains the results of applying the criteria to the XP methodology.

Table 5. Cross-Conrext evaluation criteria.

Criteria	Description	Domain Values	XP Evaluation
Performance	Defined as a function of speed, number of products, number of roles involved, and team size in each iteration.	The function is defined by the evaluators according to their preferences. We have defined it as the weighted sum of *speed*, *number of products, number of roles involved,* and *team size.*	1/4 (1/14 + 4 + 7 + 10) 5.31
Usability	Defined (based on [34, 35]) as a function of the number of *guidelines* and *roles*, and the degree of *leanness* and *domain compliance*.	The function can be defined as the sum of the relevant parts.	*This criterion is evaluated regarding the project at hand.
Completeness	Defined as a function of process completeness, coverage of umbrella activities, and specification of modeling language(s).	We have defined it as the weighted sum of the relevant criteria (equal weights have been assigned in our evaluation of XP).	1/3 (6/8 + 4/5 + 0) 0.52 (Medium)
Status	Current status of the methodology	Nascent, Building up, Active, Fading (Dead) [4]	Active
Development process	Does the methodology explain the development process?	Explicit, Implicit, No	Implicit
Modeling language	Does the methodology prescribe the use of modeling languages?	Yes, No	No
Constraints	General constraints	Any general constraints in the methodology that influences practicality.	Collective code ownership

As inferred from the table, the result of applying cross-context criteria can be calculated based on the evaluation results of the context-specific criteria. The evaluation is therefore simple and straight forward.

5. Evaluating XP using CEFAM

The last columns in Tables 1 to 5 show the results of evaluating the XP methodology using CEFAM. This evaluation is an example of the application of CEFAM in evaluating an agile methodology. Evaluators who use CEFAM can select an appropriate subset of the evaluation criteria and, if required, adapt the domain values according to their needs.

The following basic information about XP helps better understand some of the results:

- **Phases:** Exploration, Planning, Iterations to Release, Productionizing, Maintenance, and Death.
- **Roles:** Programmer, Customer, Tester, Tracker, Coach, Consultant, and Manager.
- **Products:** User stories, Metaphor, Code, Test cases, and System documentation.
- **Practices:** Planning game, Small/short releases, Metaphor, Simple design, Testing, Refactoring, Programming, Collective code ownership, Continues integration, 40-hour week, On-site customer, and Coding standards.

Evaluating XP according to CEFAM *process* criteria highlights several important issues. The development process widely recognized as the XP process is just a typical example of applying XP principles and practices. In striving to remain abstract, XP has remained a set of principles and practices brought together; the development process is therefore not explicitly and unambiguously defined and rationalized. Evaluation according to CEFAM *modeling language* criteria seems to stress XP's lack of attention to modeling and modeling-language issues.

Regarding *agility*, XP seems to be at an acceptable level, as most of the results are high. As for *usage*, XP seems to be strong as to application scope and support for umbrella activities. Moreover, XP enjoys a rich repertoire of papers, books, and experience reports. However, method tailoring is a weakness of XP, mainly because of the process's implicit and ambiguous nature, resulting in a lack of attention to process instantiation and tailoring.

Finally, *cross-context* evaluation seems to point out that although XP is rich in many aspects, as a methodology it suffers from inadequate attention to modeling and process definition. Even though this inattention has been deliberate, and has indeed succeeded to enhance the agility of the process, it has also had repercussions, manifest as lack of scalability (to name just one).

6. Conclusions and Future Work

In spite of the widespread use of agile methodologies and their ever-increasing popularity, answers to questions concerning their suitability for particular projects/domains are still difficult to find. So is the case with agile methodology engineering issues, as extensive and precise scrutiny of existing methodologies is a prerequisite not yet achieved. In order to address these challenges and requirements, developers, managers and method engineers need an appropriate evaluation framework that provides detailed evaluations of agile methodologies. Our proposed Comprehensive Evaluation Framework for Agile Methodologies – CEFAM – aims at addressing this need.

CEFAM is a comprehensive evaluation framework, aiming at covering all the different aspects of agile methodologies, especially focusing on issues that are essential in agile process enactment and method engineering. The hierarchical and mostly quantitative nature of the proposed evaluation framework enhances its usability and provides results that are precise enough to be used by project managers and method engineers for methodology selection and construction. Since CEFAM has been designed to satisfy the evaluation framework meta-criteria of [1], every effort has been made to ensure that all the qualities that are considered desirable in a methodology evaluation framework are duly achieved.

An important feature of CEFAM is the quantitative nature of many of the criteria. However, there were cases where providing a quantitative version for a criterion was not achievable, mainly because it required research that was beyond the scope of this paper. In such cases, relevant parameters have been introduced for the criterion, thus facilitating evaluation while leaving detailed appraisal to the evaluator.

Future research in this regard will be mostly focused on refining the framework through applying it to other agile methodologies (preferably based on empirical feedback acquired from real project situations), and defining more detailed quantitative criteria. The research can be expanded to method engineering, mainly through fusing CEFAM into a Situational Method Engineering (SME) process [19]; such processes are aimed at the adaptation/construction of

bespoke methodologies according to the particulars of the project situation at hand.

7. Acknowledgment

We wish to thank Mr. Hamed Yaghoubi Shahir for presenting this paper at SEW'08. We also extend our gratitude to the ITRC research center for partial sponsorship of this research.

8. References

[1] M. Taromirad, R. Ramsin, "An Appraisal of Existing Evaluation Frameworks for Agile Methodologies", *In Proceedings of the 15th IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS'08)*, Northern Ireland, 2008, pp. 418-427.

[2] P. Lappo, H. C.T. Andrew, "Assessing Agility", *Lecture Notes on Computer Science*, Springer-Verlag, Germany, 2004, pp. 331-338.

[3] D. Turk, R. France, B. Rumpe, "Assumptions Underlying Agile Software Development Processes", *Journal of Database Management*, Idea Group Inc., October-December 2005, pp. 62-87.

[4] P. Abrahamsson, O. Salo, J. Ronkainen, J. Warsta, *Agile Software Development Methods: Review and Analysis*, VTT Publication, Finland, 2002.

[5] P. Abrahamsson, J. Warsta, M. Siponen, J. Ronkainen, "New Directions on Agile Methods: A Comparative Analysis", In *Proc. of the 25th International Conference on Software Engineering (ICSE'03)*, Oregon, 2003, pp. 244-254.

[6] J. Koskela, *Software Configuration Management in Agile Methods*, VTT Publication, Finland, 2003.

[7] L. Williams, W. Kerbs, L. Layman, A. Anton, "Toward a Framework for Evaluating Extreme Programming", In *Proc. of the 8th International Conference on Empirical Assessment in Software Engineering (EASE 04)*, Edinburgh, 2004, pp. 11-20.

[8] E. Germain, P. Robillard, "Engineering-based Processes and Agile Methodologies for Software Development: a Comparative Case Study", *The Journal of Systems and Software*, Elsevier, February 2005, pp. 17-27.

[9] A. Qumer, B. Hendersson-Sellers, "Comparative Evaluation of XP and Scrum Using the 4D Analytical Tool (4-DAT)", In *Proceedings of the European and Mediterranean Conference on Information Systems (EMCIS)*, Spain, 2006.

[10] M. Pikkarainen, U. Passoja, "An Approach for Assessing Suitability of Agile Solutions: A Case Study*", In Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2005)*, UK, June 2005, pp. 171-179.

[11] D. Turk, R. France, B. Rumpe, "Limitations of Agile Software Processes", *In Proceedings of the 3rd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP 2002)*, Italy, May 2002, pp. 43-46.

[12] K. Beck et al., "Manifesto for Agile Software Development", Available at http://www.agilemanifesto.org.

[13] Agile Alliance, "Agile Principles", Available at http://agilealliance.org.

[14] K. Conboy, B. Fitzgerald, "Toward a Conceptual Framework of Agile Methods: A Study of Agility in Different Disciplines", *In Proceedings of the 2004 ACM workshop on Interdisciplinary software engineering research*, CA, USA, 2004, pp. 37-44.

[15] B. Boehm, R. Turner, "Using Risk to Balance Agile and Plan-Driven Methods", *Computer*, IEEE, June 2003, pp. 57-66.

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

[17] U. Frank, "A Comparison of two outstanding Methodologies for Object-Oriented Design", *FIT.CSCW*, 1992.

[18] R. Ramsin, "Evaluation of Object-Oriented Software Development Methodologies", *In Proceedings of the 1st Computer Society of Iran Computer Conference (CSICC'95),* Iran, 1995, pp. 40-50.

[19] R. Ramsin, "The Engineering of an Object-Oriented Software Development Methodology", *Ph.D. Thesis*, University of York, UK, 2006.

[20] A. Sturm, O. Shehory, "A Framework for Evaluating Agent-Oriented Methodologies", In *Proc. of the 5th International Bi-Conference Workshop on Agent-Oriented Information Systems*, 2003, pp. 94-109.

[21] P. Cuesta, A. Gomez, J. Gonzalez, F. Rodrıguez, "A Framework for Evaluation of Agent Oriented Methodologies", *In Proc. of The Conference of the Spanish Association for Artificial Intelligence (CAEPIA)*, Spain, 2003.

[22] K. H. Dam, "Evaluating and Comparing Agent-Oriented Software Engineering Methodologies", *MS Thesis*, RMIT University, Australia, 2003.

[23] B. George, "Analysis and Quantification of Test Driven Development Approach", *MS Thesis*, North Carolina State University, USA, 2002.

[24] M. Alshayeb, W. Li, "An empirical study of system design instability metric and design evolution in an agile software process", *The Journal of Systems and Software*, Elsevier, March 2004, pp. 269-274.

[25] B. Rumpe, A. Schroder, "Quantitative Survey on Extreme Programming Projects", *In Proceedings of the 3rd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP 2002)*, Italy, May 2002, pp. 95-100.

[26] D. Sato, D. Bassi, M. Bravo, A. Goldman, F. Kon, "Experiences Tracking Agile Projects: an Empirical Study", *Journal of the Brazilian Computer Society*, 2007.

[27] D. Sato, A. Goldman, F. Kon, "Tracking the Evolution of Object-Oriented Quality Metrics on Agile Projects", 2007.

[28] M. Rossi, S. Brinkkemper, "Complexity Metrics for Systems Development Methods and Techniques", *Information Systems*, Elsevier, April 1996, pp. 209-227.

[29] F. Garcı´a, M. Piattini, F. Ruiz, G. Canfora, C. A. Visaggio, "FMESP: Framework for the modeling and evaluation of software processes", *Journal of Systems Architecture*, Elsevier, August 2006, pp. 627-639.

[30] G. Canfora, F. Garcı´a, M. Piattini, F. Ruiz, C. A. Visaggio, "A family of experiments to validate metrics for software process models", *The Journal of Systems and Software,* Elsevier, December 2004, pp. 113-129.

[31] S. Tasharofi, F. Ghasemi, "Evaluation Criteria for Agile Methodologies", Research Report, Sharif University of Technology, Iran, 2007.

[32] R. S. Pressman, *Software Engineering: A Practitioner's Approach 5th ed*., McGraw-Hill, 2001.

[33] A. Cockburn, "Selecting a Project's Methodology", IEEE Software, IEEE, July/August 2000, pp. 64-71.

[34] B. Boehm, et al. *Software cost estimation with COCOMO II* (with CD-ROM). Englewood Cliffs, NJ:Prentice-Hall, 2000.

[35] *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*, Fourth Edition, Project Management Institute, 2008.