2010 34th Annual IEEE Computer Software and Applications Conference

Towards a General Framework for Evaluating Software Development Methodologies

Shokoofeh Hesari, Hoda Mashayekhi, Raman Ramsin
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
hesary@ce.sharif.edu, mashayekhi@ce.sharif.edu, ramsin@sharif.edu

***Abstract*—It has become essential to scrutinize and evaluate software development methodologies, mainly because of their increasing number and variety. Evaluation is required to gain a better understanding of the features, strengths, and weaknesses of the methodologies. The results of such evaluations can be leveraged to identify the methodology most appropriate for a specific context. Moreover, methodology improvement and evolution can be accelerated using these results. However, despite extensive research, there is still a need for a feature/criterion set that is general enough to allow methodologies to be evaluated regardless of their types.**

We propose a general evaluation framework which addresses this requirement. In order to improve the applicability of the proposed framework, all the features – general and specific – are arranged in a hierarchy along with their corresponding criteria. Providing different levels of abstraction enables users to choose the suitable criteria based on the context. Major evaluation frameworks for object-oriented, agent-oriented, and aspect-oriented methodologies have been studied and assessed against the proposed framework to demonstrate its reliability and validity.

Keywords- software development methodology; evaluation framework; process feature; evaluation criterion.

I. INTRODUCTION

With the emergence and widespread application of software systems in recent decades, it has become necessary to introduce software development processes targeting different features and paradigms. The number and variety of software processes and software development methodologies has made it difficult to select a methodology for a specific project, or to construct the appropriate methodology through assembling method chunks. Methodology evaluation has hence become an essential task.

Apart from the research conducted on analysis and evaluation of software development methodologies and processes, there is still a need for a general multi-aspect framework that facilitates the evaluation of methodologies of different types and paradigms (function-oriented, object-oriented, aspect-oriented, etc.). Indeed, the lack of general criteria that target different features is a major deficiency of existing evaluation frameworks.

On the other hand, a potential problem in devising a general and multi-aspect evaluation framework is defining a large set of requirement features, as these may be difficult for the user to understand. This may in turn cause the framework to lose its applicability. When comparing methodologies that belong to different contexts, multiple sets of criteria at different levels of detail may be required. Hence, a suitable evaluation framework should offer the possibility of extracting and tailoring a desired subset of criteria that provides the appropriate degree of precision. This can facilitate the selection of methodologies by the user. However, when no specific context is involved, only the general and main features of methodologies can be compared [1]. To overcome these problems, an appropriate solution is to categorize the set of features at different levels of abstraction. A hierarchical set of criteria is therefore preferable, as exemplified by the ISO-9126 standard [2]. By presenting the features from the user's viewpoint, the framework gains in understandability and applicability.

We propose a general methodology evaluation framework that can be instantiated and refined to fit specific paradigms and evaluation situations. The framework has been defined through studying, abstracting and unifying methodology features and existing evaluation frameworks. To this aim, methodologies and their general features have first been studied. An analytical assessment of existing evaluation frameworks – targeting object-oriented, agent-oriented and aspect-oriented methodologies – has then been carried out, thereby identifying their capabilities and limitations. Lastly, the target multi-aspect framework has been delineated by defining, abstracting, and structuring a set of hierarchical evaluation criteria so that the ultimate goals of "generality" and "refinability" are attained. By highlighting the features, strengths and weaknesses of existing methodologies and method chunks, the proposed evaluation framework can facilitate the selection of methodologies, selection and assembly of method chunks, extension and tailoring of methodologies, and evaluation of other evaluation frameworks.

The rest of this paper is structured as follows: Section 2 describes the proposed features and the evaluation framework along with the assumptions and methods used; Section 3 provides a brief survey on existing evaluation frameworks for object-oriented, agent-oriented, and aspect-oriented methodologies along with an assessment on the generality of the proposed framework; and the final section presents the conclusions, and provides ideas for furthering this research.

0730-3157/10 $26.00 © 2010 IEEE
DOI 10.1109/COMPSAC.2010.69

II. PROPOSED EVALUATION FRAMEWORK

Applying evaluation frameworks to methodologies offers several advantages. Evaluation can aid in understanding particular aspects of a methodology. It also acts as a means of comparing methodologies, and thus helps the user choose among multiple methodologies according to desired goals. Moreover, methodology enhancement and improvement can be accelerated using the results of evaluation [3].

According to [4], an evaluation framework consists of three components: A list of feature requirements, a method of scoring the features in the methodologies targeted, and a set of guidelines for applying the evaluation framework. The first and most important component is a list of feature requirements for evaluation. It is essential that this feature list addresses the various parameters of software development processes, highlighting the similarities, differences, features, and application contexts of methodologies. A common reference is needed to evaluate and analyze the features belonging to different methodology types. This *reference model* will serve as a basis for assessing the maturity of target methodologies, and their applicability in a specific software development project. This model should be a minimized and consistent superset of the features found in all evaluation frameworks. Such a model is extensively used in meta-model-based evaluation techniques. For example, an evaluation meta-model is presented in [5] that is a superset of features found and expected in object-oriented methodologies. In order to attain a suitable reference model, it is first necessary to crisply define what a Software Development Methodology (SDM) is.

Multiple definitions are proposed for an SDM. The definition provided by OMG [6] is widely accepted. According to this definition, a software development methodology is composed of two main components: A modeling language which consists of a set of modeling conventions (syntax and semantics), and a process which provides a guide as a sequence of software production activities, describes the artifacts that should be produced by the modeling language, manages and directs team efforts, and provides criteria for monitoring and assessing project activities and outputs. According to this definition, the process is the dynamic and behavioral component of the methodology, which handles the technical production and managerial sub-processes. As a result, the process includes the stages, procedures, rules, techniques and tools that are defined by the methodology, and also provides guidelines on documentation and management [7].

Based on this definition, a software development process should possess certain features in order to be considered a methodology. We designate these features as *generic methodology features*. Evaluation of these features can lead to an appraisal of the methodology's maturity level and its applicability in a specific context, regardless of the methodology type. The distinctive concepts which are specific to a specific type of methodology are addressed as *Type-specific methodology features.* We have thus categorized methodology features according to the following conceptual hierarchy, based on user viewpoint (Fig. 1):

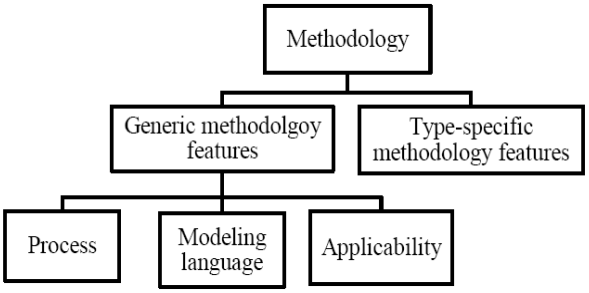

Figure 1. Main hierarchy of features

- Generic methodology features
  - Modeling language features
  - Process features
  - Applicability features
- Type-specific methodology features

To determine the requirements pertaining to a feature, the feature is broken down into sub-features. Conceptual commonalities exist among the sub-features under a super-feature. These commonalities can be leveraged to discover the corresponding sub-features. This method prevents the forming of a complex and large set of requirements, and supplies semantics for the set of features.

In order to score the features, we define a set of criteria corresponding to each feature's requirements. These criteria cover the proposed requirements of the methodology. To define each criterion, the viewpoint and the level of detail should be specified. Maintaining the applicability of the evaluation requires the criteria to be expressed from the user viewpoint.

Some of the criteria may have conceptual commonalities with each other; therefore, they can be inserted under a common super-criterion. It is permissible for a criterion to be a sub-criterion of more than one super-criterion. A similar method is employed in the software product evaluation standard ISO-9126 [2].

The super-criteria may be directly involved in the evaluation. However if direct evaluation is impractical and/or more detail is required, evaluation of a super-criterion can be delegated to its sub-criteria. In this case, an overall assessment of the sub-criteria can be used to estimate the score of the super-criterion. A motivation for this delegation can be the existence of techniques, procedures, and other utilities for handling a requirement. Thus, the fulfillment of a requirement is dependent on employing the corresponding utilities introduced by the methodology. This hierarchical structure is a means of regulating a complex and large set of criteria. A weight can be assigned to each evaluation criterion, reflecting the relative importance of the criterion. This weight is resolved based on methodology type and user decision. A sub-criterion which is common among multiple super-criteria can have different weights for each of its instances.

The relationships that exist between a methodology and its related concepts are shown in Fig. 2. Methodology features are explained in detail throughout the rest of this section.

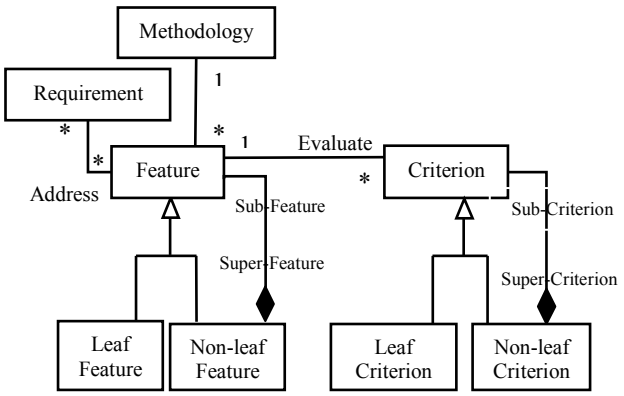

Figure 2. Relationships between methodology and other concepts

A. *Modeling language features*

A modeling technique is a set of models that depict a system at different levels of abstraction and describes its different aspects [8]. In this section, we present the super-and sub-criteria that are related to modeling languages. Fig. 3 shows the corresponding hierarchy of criteria. The criteria are briefly described below:

1. **Ease of**
   - **Understanding**: understandability of a modeling technique [9].
   - **Usage**: usability of a modeling technique.[9]
2. **Preciseness** [9]**:** the syntax and semantics of a modeling language must be precise enough to avoid ambiguity.
3. **Power of language:**
   - **Formalism:** existence of a formalization aspect for symbolizing the semantics.
   - **Supported views:** Coverage of structural, behavioral and functional views in the modeling techniques.
   - **Support for model transformation logic:** provision of logic for transformation between models or transformation of models to code [9, 10].
4. **Model complexity management:**
   - **Modularity:** specification of a model in an iterative-incremental manner; that is, when new requirements are added, existing specifications can be used but should not be altered [11].
   - **Handling model inconsistencies:** provision of techniques for handling inconsistencies [12].
5. **Expressiveness:**
   - **Static and dynamic aspects:** capability of expressing both static and dynamics concepts [9].
   - **Physical architecture of systems:** capability of expressing the system's architectural design [11].
   - **Constraints of systems:** capability of representing system constraints [11].
   - **Defining user interface:** capability of representing the user interface in models [11].

B. *Process features*

The process has two main roles [10]: It manages and directs the development from analysis to implementation, and it enables improvement traceability by defining deliverables and milestones. We therefore divide the *Process* feature into three sub-features: *lifecycle*, which focuses on the development lifecycle; *management aspects*, which refer to management activities throughout the lifecycle; and the *development context*, which focuses on the development context(s) supported by the methodology (Fig. 4). In the following subsections, we present the features and the criteria for each sub-feature.

1) Lifecycle

As observed in Fig. 5, we have analyzed the lifecycle from a method engineering viewpoint, focusing on its three types of components: work units, products, and roles.

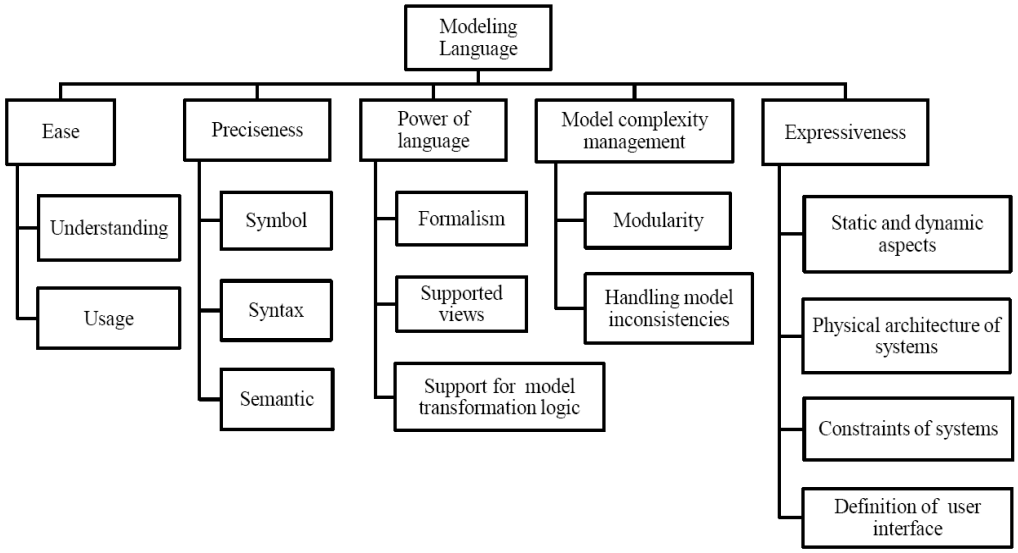

Figure 3. Modeling language hierarchy

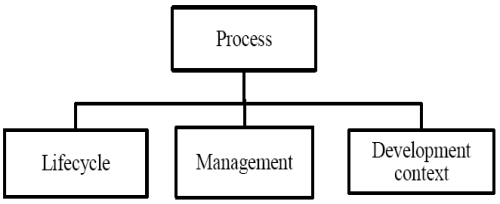

Figure 4. Process hierarchy

The sub-features and corresponding criteria are listed below:

1. **Work Unit:**
   - **Coverage of generic phases:** whether the development process covers the generic phases of software development (Requirements, Analysis, Design, Implementation, and Test) [8].
   - **Transition between phases**
     i. **Smooth transition:** whether transition between phases is smooth [12].
     ii. **Seamless transition:** whether there is any semantic gap between the artifacts produced by the phases [12].
   - **Kind of lifecycle:** describing the kind of lifecycle model that is applied in the development process (iterative, incremental, cascade, etc.) [8].
   - **Workflow:** constituents of the lifecycle [9].
2. **Product [12]:**
   - **Adequacy:** whether the development process produces the products that are related to the phases.
   - **Consistency:** whether the products complement each other with minimum overlapping.
   - **Supported view:** specifying the generic view that the products support (structural, behavioral, or functional).
   - **Abstraction levels:** the granularity/abstraction levels at which the products are presented (system, package, component, object, etc. – at analysis, design, or implementation levels).
   - **Tangibility/Testability/Visibility:** whether products are tangible, testable, and understandable.
   - **Appropriate Documentation:** support for proper documentation throughout the development lifecycle.
3. **Role:**
   - **User involvement:** whether the users are involved in the process through specially defined roles [12].
   - **Roles specification:** whether development roles are specified in the process.

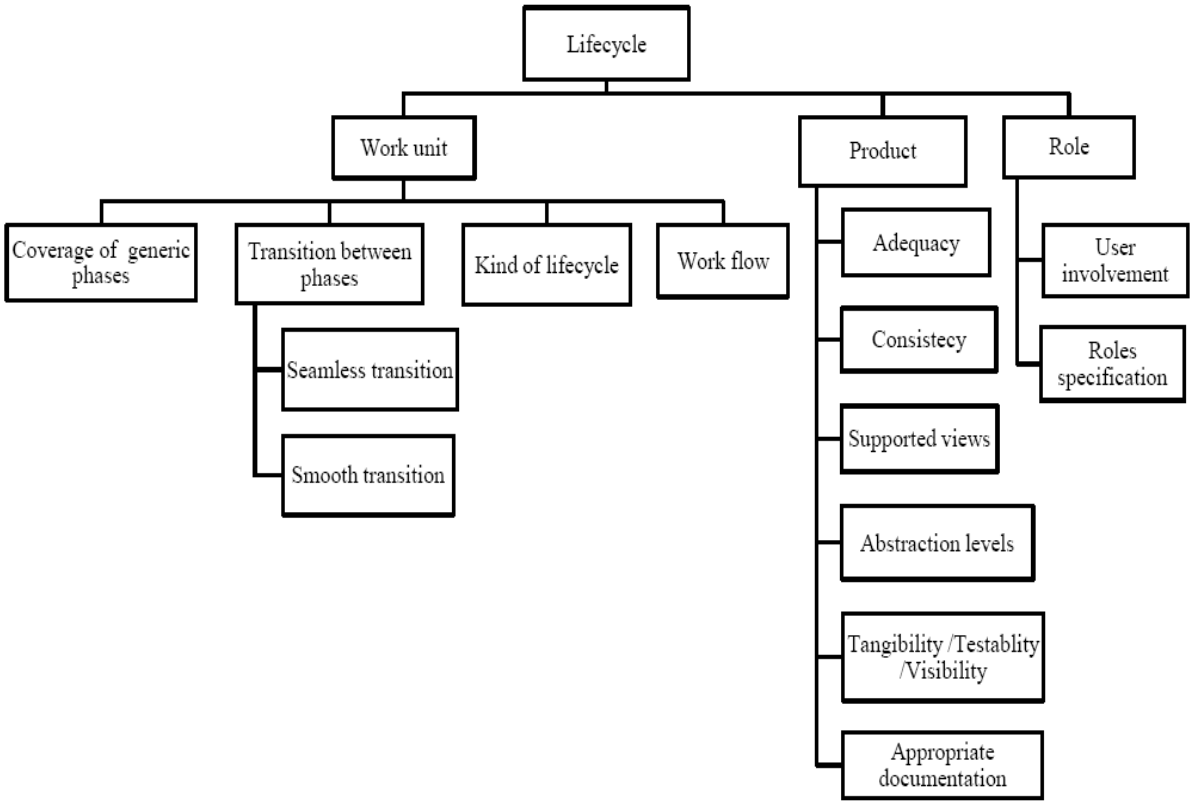

Figure 5. Lifecycle hierarchy

2) Management

Management features address the support that a methodology provides for management [13]. The detailed features are presented in Fig. 6. In what follows, the sub-features are briefly described (as adapted from [14]):

1. **Risk management:** avoidance, monitoring and management of risk to assist the project team in developing a strategy for handling risks.
2. **People management:** practices such as team formation, training, performance management, etc.
3. **Quality management:**
   - **Quality control:** practices for ensuring quality in products, such as reviews and the use of tools.
   - **Quality assurance:** reporting practices for assuring the effectiveness and completeness of quality control activities.
4. **Configuration management:** activities for change management throughout the development lifecycle.
5. **Project scheduling:** distribution of the effort estimated across the planned project duration by allocating the effort to specific tasks to make the best use of the available resources, including time.

3) Development context

Development context features focus on features that specialize the context of the development effort from a user viewpoint.

This feature is subcategorized as shown in Fig. 7. Brief descriptions are provided below for the features and the corresponding criteria:

1. **Ease of**
   - **Understanding:** understandability of the development process.
   - **Usage:** usability of the development process in its intended context.

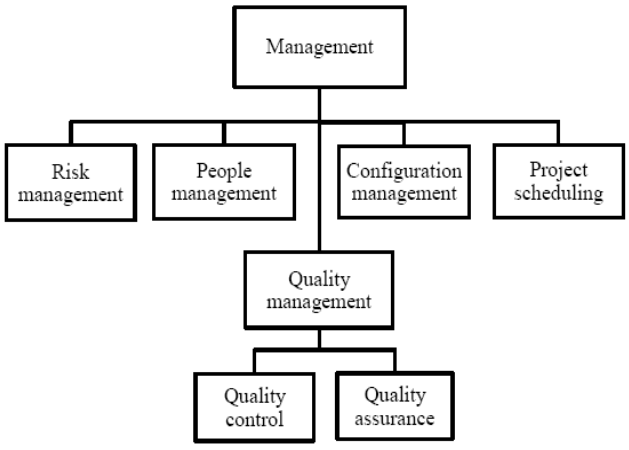

Figure 6. Management hierarchy

2. **Efficiency [2]:**
   - **Time:** the balance between the development throughput, and the time that its process consumes.
   - **Resources:** the amount of resources, including human and financial, that is used with respect to the throughput.
3. **Precision:**
   - **Traceability:** whether the artifacts can be traced to the requirements or to real world concepts.
   - **Formalism:** at the process level.
   - **Well-definedness:**
     i. **Expressiveness:** the ability to define the process without ambiguity [12].
     ii. **Rationality:** logical appeal of the process [12].
     iii. **Completeness:** A complete definition must include rigorous explanations for all aspects of the methodology, including work units, products, and people.
4. **Maintainability:**
   - **Modularity:** the ability to preserve the parts corresponding to components from side effects [1].
   - **Reusability:** The ability to reuse the process in multiple applications.
   - **Testability:** possibility and practicality of phase verification against the outcomes of previous phases, and product validation against user requirements.
5. **Complexity Management:**
   - **Evolvability:** the ability to increasingly improve the system's functional and nonfunctional aspects.
   - **Extensibility:** the ability to expand the system to a certain degree [9].
   - **Promoting complex architectures:**
     i. **Distribution:** support for modeling and implementation of components supporting distributed functionality.
     ii. **Integrity:** provision of an integrated architecture that conforms to model semantics .

C. Applicability features

Another important aspect is the evaluation of the methodology as to its applicability in a software development project. Applicability of a methodology is an essential characteristic which is evaluated with the following super-criteria: *pragmatics, marketability,* and a*pplication constraints*. Evaluation of each of these criteria can be delegated to lower-levels sub-criteria. The classification hierarchy is presented in Fig. 8. Brief descriptions of the criteria are provided below:

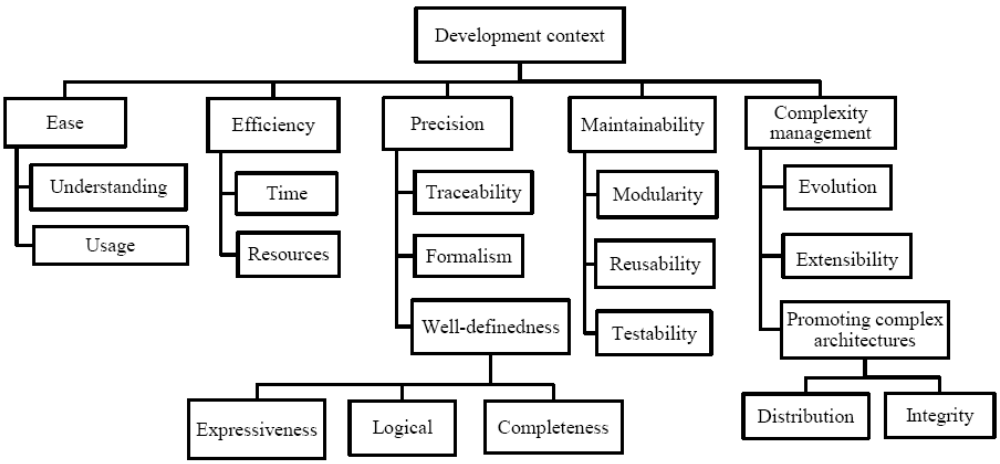

Figure 7. Development context hierarchy

1. **Pragmatics:**
   - **Adaptability:**
     i. **Size and complexity:** determines the size of the methodology and its complexity.
     ii. **Criticality:** loss due to the impact of defects [12].
     iii. **Scalability:** adaptability of the methodology to different project sizes [9].
   - **Extant resources:**
     i. **Available information:** availability of documents, instructions, etc [9, 10 and 12].
     ii. **Tools:** availability of resources and CASE tools that support the methodology [9, 10 and 12].
   - **Required resources:**
     i. **Team skills:** adaptability of development team skills with the methodology.
     ii. **Platform suitability:** adaptability of resources including extant middleware, libraries, and tools.
2. **Marketability:**
   - **User satisfiability:** the degree of end-user satisfaction with respect to the performance and cost of the delivered product [9].
   - **Development team satisfiability:** satisfaction of individuals as to their specific roles.
3. **Application constraints:**
   - **Legal constraints:** such as contract types and business culture.
   - **Technical constraints:** such as programming languages and platforms [12].
   - **Management constraints:** such as management culture and approaches [12].

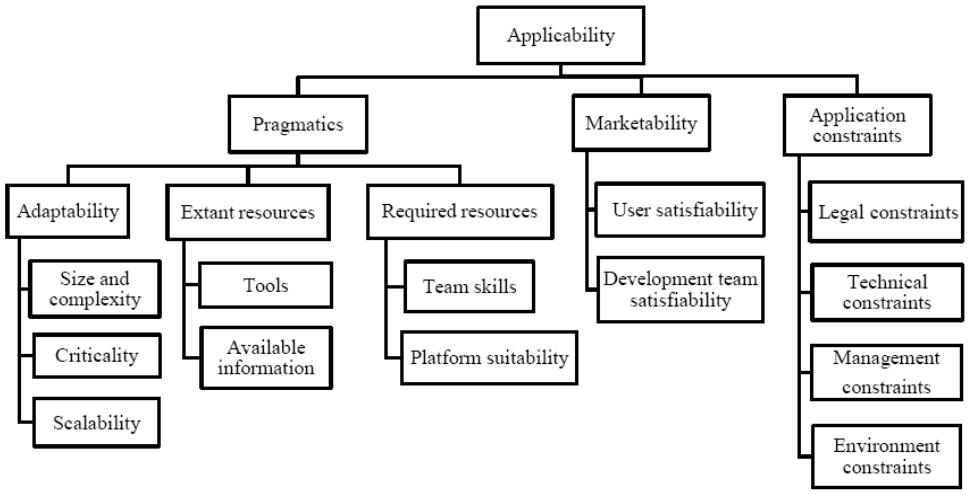

Figure 8. Applicability hierarchy

- **Environment constraints:** constraints that are influenced by the environment, such as physical layout [12], and location of development teams (also called geographical constraints [12]).

D. Type-specific methodology features

Fulfilling the requirements and avoiding undesirable events are the main concerns in a methodology, based on which the behavior of the methodology is uniquely identified. From this point of view, each methodology introduces its own set of identifying concepts. A Concept is an abstraction or perception derived from a specific case in the problem domain, which can be a property, capability, feature, or entity [13]. With the introduction of concepts in a methodology, it may be required that the evaluation of a general super-criterion be delegated to other criteria pertaining to the type of the methodology. In other words, because of possible emphasis on a general requirement in a methodology, special procedures, rules, techniques, or roles may have to be considered. We are therefore forced to define new sub-criteria for the evaluation of a general criterion, and delegate the evaluation to these sub-criteria. We refer to these as *philosophy* criteria. The special concepts provided by a methodology type may introduce new requirements, and should therefore be evaluated separately. For this reason, new criteria are defined to evaluate the methodology's type-specific concepts. We refer to these as concept-specific criteria. The taxonomy described above is shown in Fig. 9.

In order to exemplify and validate this taxonomy, we assess agent-oriented methodologies against this categorization. Agent-oriented methodologies have been applied to complex systems; promoting complex architectures is therefore a philosophy of agent-oriented methodologies. To this end, these methodologies propose special concepts, most of which can be classified under general feature requirements. In addition to these concepts, the main methodology-specific concept proposed in agent-oriented methodologies is the *Agent*; this concept should be independently evaluated based on its feature requirements. Such criteria have been addressed in most agent-oriented evaluation frameworks. Fig. 10 shows the relevant hierarchy.

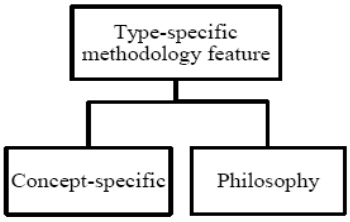

Figure 9. Type-specific methodology features hierarchy

III. Evaluation Frameworks for Specific Types of Methodologies

In this section, existing evaluation frameworks for object-oriented, aspect-oriented, and agent-oriented methodologies are studied and assessed against our proposed evaluation framework.

A. Evaluation of object-oriented methodologies

Object-oriented software development has gained in popularity in recent years. Since objects provide a more logical and natural correspondence with real world concepts, object-oriented processes are deemed better suited for conceptual modeling. Inheritance and encapsulation promote reusability of concepts and components. Moreover, using an object-oriented model facilitates integration in organizations.

Object-oriented methodologies model the real world with objects, whereas their predecessors concentrate on functions, and separate process from data. Therefore, one cannot simply map all the criteria defined for non-object-oriented methods to object-oriented concepts. Some of the research efforts conducted on comparing object-oriented methodologies focus on the complete methodology, while others concentrate on only some aspects of the methodology.

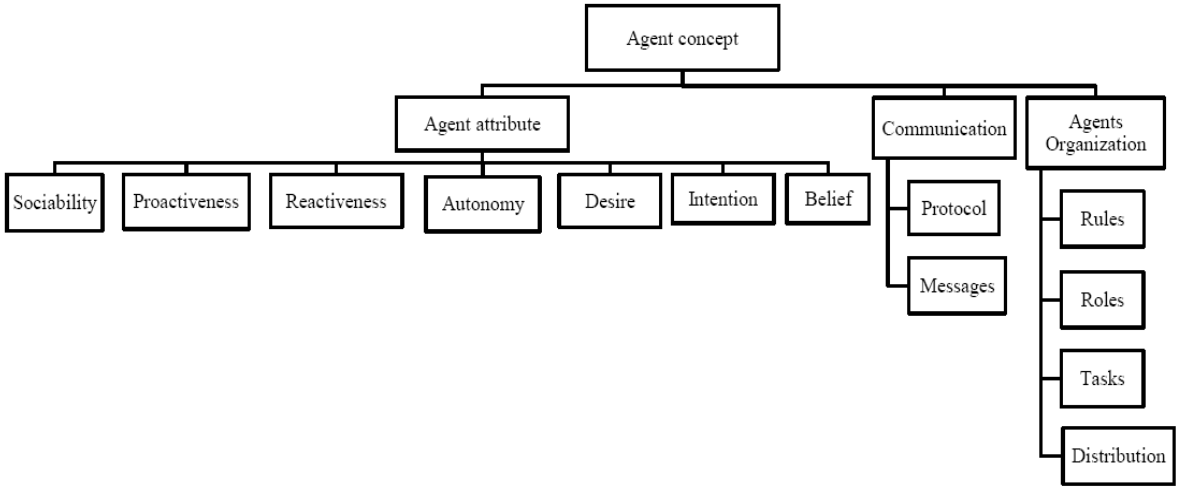

Figure 10. Agent-oriented methodology features hierarchy

The popularity and flexibility of feature analysis as a strong evaluation approach has placed this method among the most widely employed evaluation techniques. Some proposals have made an effort to provide an ideal general object-oriented framework, while others target a specific context. These frameworks address the main components of an object-oriented methodology, such as concepts, processes, tools, and practical features.

An overview of 30 object-oriented methodologies is provided in [1], based on which a framework has been proposed for comparing these methodologies. As no specific context is considered, only general and main features of the methodologies are compared. This research classifies its criteria into three classes of static, behavioral, and dynamic criteria. The static aspect analyzes the permissible states or structures. The behavioral aspect considers the expected functionalities of the system, and the dynamic aspect assesses the behavior or the system during its lifetime.

Packard compares five object-oriented methodologies, considering a set of criteria which are classified into concepts, models, process, and pragmatics [10].

Evaluation using a meta-model builds a common reference for comparing different methodologies. This reference is usually constructed based on a composition of various features of the target methodologies. Hong et al. [5] propose a formal model for comparing object-oriented methodologies. A formal representation of each methodology is built using two models; meta-process and meta-data. This uniform presentation allows for global comparison, and prevents errors due to wrong interpretations of the methodology. This research overlooks the comparison of guidelines and rules provided by different methodologies. Also, due to the limitations of the entity-relationship model (used for metamodeling), some object-oriented concepts are difficult to display, and this can adversely affect comparison accuracy. In addition, issues such as methodology guidelines for designing a promoted software system to maximize the advantages gained from object-oriented technologies (such as reusability, maintainability, and changeability) are not considered.

Process output is analyzed and assessed using multiple criteria. For example, product quality or complexity is appraised. This is actually a type of quantitative evaluation [13]. Object-oriented methodologies have the largest amount of quantitative evaluation criteria among SDMs. According to [13], this kind of evaluation is of two types: formal examination and metric-based techniques.

Various sets of metrics are provided for object-oriented design, and especially for object-oriented programming [15, 16]. Two perpendicular criteria vectors have been proposed in [17], called granularity and category. The category vector is introduced in six groups, namely design, size, complexity, reusability, productivity, quality and general procedures. The granularity vector contains method, class, and system, and is perpendicular to the category vector.

In table 1, an assessment of current evaluation approaches targeting object-oriented methodologies is presented by comparing them to our proposed framework.

TABLE I. ASSESSMENT OF OBJECT-ORIENTED FRAMEWORKS

	Generic features					Specific features
		Process				
	Modeling language	Lifecycle	Management aspects	Development context	Applicability	Concept-specific
Frank [1]	✓	P	-	P	-	✓
Hong [5]	✓	✓	-	-	-	✓
Packard [10]	✓	✓	-	P	P	✓

✓: acceptable coverage, **P:** partial coverage, **-:** no coverage

B. Evaluation of aspect-oriented methodologies

Aspect-oriented software development is a recent but fast-growing area. Aspect-orientation considers modularization, encapsulation, and crosscutting concerns as its main foci. Special concepts, along with the corresponding terminology, are leveraged in aspect-oriented methodologies, thus necessitating separate evaluation. Crosscutting concerns refer to those sets of concerns that cannot be effectively modularized by object-oriented techniques. These concerns typically originate from non-functional requirements. The separation of these concerns in modules means that they should be later connected with composition mechanisms. Moreover, a traceability mechanism is required to map the concerns betweens different phases.

A common architecture reference for aspect-oriented modeling is presented in [18], which distinguishes main aspect-oriented subjects from aspect-oriented programming or aspect-oriented modeling. Among the benefits of this method are the possibility of creating a framework of evaluation criteria, mapping different aspect-oriented methods to each other, and using the architecture as a meta-model for designing a new integrated modeling language.

The research performed on evaluation of aspect-oriented methodologies typically focuses on one of the activities in the system lifecycle [19, 20]. In [20], an attempt is made to study aspect-oriented design methods, and locate them in the generic software lifecycle. It introduces evaluation criteria for assessing aspect-oriented design methods, classifying them and analyzing their effects on software quality factors.

The artifacts produced during the lifecycle are focused upon in [21]. It proposes a systematic way of quantitative evaluation of aspect-oriented artifacts produced during design and implementation. It introduces a set of criteria, a set of rules, and an evaluation tool.

Evaluation of aspect-oriented features along system development phases has also attracted the attention of the research community. Traceability relations can be used to define crosscutting concerns, as proposed by the method introduced in [22]. It constructs a dependency matrix to capture the relationships among different levels, such as concerns and their representations. It formalizes the definition of crosscutting concerns, and detaches it from other concepts such as scattering and tangling. Although this

method is applicable in different phases of system development, its scalability and its applicability to different types of traceability relations should be further analyzed. An assessment of modularization in aspect-oriented design is presented in [23].

Object-oriented criteria can be used to compare object-oriented and aspect-oriented methods, as proposed in [24]. This research concludes that better modularization of the system does not necessarily improve other features of the system, such as maintainability and reusability. These features should therefore be analyzed independently.

Table 2 demonstrates the results of evaluating a number of evaluation methods and frameworks targeting aspect-oriented methodologies against our proposed framework.

C. Evaluation of agent-oriented methodologies

The concept of *Agent*, with properties such as autonomy, responsiveness, correctness, logicality, and interactivity, is considered a strong tool for developing distributed systems [8]. Due to increasing interest in agent-oriented applications, various agent-oriented methodologies have been proposed [13]. However, only a handful of them are mature enough to address the industrial requirements of agent-oriented systems development. To avoid building methodologies from scratch, agent-oriented methodologies are constructed by extending existing methodologies based on agent aspects. These extensions are commonly applied to two types of methodologies: object-oriented and knowledge-engineering.

Many of the proposals for evaluating agent-oriented methodologies use feature analysis. However, some of them have introduced quantitative and qualitative criteria, and use novel approaches. For example, criteria are arranged in a tree-like structure in [25], and they are assigned weights based on the weights of their children. Other than feature analysis, descriptive methods are also applied in some research efforts. For example, the method introduced in [26] proposes a challenge exampler modeling approach for evaluation. In this method, to gain a better understanding of the strong and weak points of different methodologies, an exampler of standard instances is defined and managed. No framework has been proposed in this research.

TABLE II. ASSESSMENT OF ASPECT-ORIENTED FRAMEWORKS

	Generic features					Specific features
		Process				
	Modeling language	Lifecycle	Management aspects	Development context	Applicability	Concept-specific
Schauerhuber [19]	✓	-	-	✓	P	✓
Berg [22]	-	-	-	-	-	✓
Figueiredo [21]	-	-	-	P	-	✓
Op de beeck [20]	✓	P	-	✓	-	✓

✓: acceptable coverage, **P:** partial coverage, **-:** no coverage

Most agent-oriented evaluation frameworks have used feature-based evaluation and have chosen a similar set of criteria. They differ in specific details, such as conceptual classification of the proposed criteria, and the evaluation techniques that are applied for criteria assessment. The criteria are often presented in four general categories: *Concepts, Modeling techniques and symbols, Process,* and *Pragmatics*. A minority of the frameworks, such as [13], have considered *supporting software engineering* and *Marketability* categories too. In frameworks that solely contain these categories, assessing the maturity of methodologies is not straightforward.

Categorization of criteria into the three general aspects of methodology, development features, and methodology-specific features is observed in [8, 11, and 27]. In this regard, the framework introduced in [8] categorizes the criteria in five classes: Development Process criteria, which focus on the general aspects of the methodology and other aspects relevant to the stages of system construction; Model View criteria, which evaluate techniques and diagrams for systems modeling; Agent criteria, which evaluate the features and characteristics of agents; Additional Features Modeling criteria, which assess how special features are supported; and Documentation criteria, which focus on issues related to documenting the products.

As another example, criteria are classified into two categories in [28]: *Software Engineering Evaluation Criteria* and *Agent-Based System Characteristics*. However, these criteria have been adapted with the classification proposed in [8]. Sudeikat et al. [27], aiming at a flexible framework, have categorized their proposed criteria as *platform-dependent* and *platform-independent*.

Table 3 demonstrates the results of assessing a number of evaluation frameworks targeting agent-oriented methodologies against the proposed framework.

IV. CONCLUSIONS

Methodology evaluation is an important means of selecting the appropriate methodology in a software project, or selecting method chunks for constructing a methodology.

TABLE III. ASSESSMENT OF AGENT-ORIENTED FRAMEWORKS

	Generic features					Specific features
		Process				
	Modeling language	Lifecycle	Management aspects	Development context	Applicability	Concept-specific
Cuesta [8]	✓	✓	-	-	P	✓
Dam [13], Akbari [9]	✓	✓	P	P	P	✓
Cernuzzi [25]	✓	✓	P	P	-	✓
Yu [26]	-	P	P	✓	P	✓

✓: acceptable coverage, **P:** partial coverage, **-:** no coverage

An evaluation framework acts as an essential guide for methodology improvement and evolution. The existence of a general evaluation framework, with criteria that are adaptable according to the context, can simplify the evaluation procedure and make the results more accurate and reliable. The general evaluation framework proposed herein is composed of features/criteria that fulfill the above requirements. By assessing different evaluation frameworks for object-oriented, aspect-oriented, and agent-oriented methodologies against our proposed framework, it was observed that the majority of the available evaluation frameworks lack a general set of criteria for methodology evaluation, and that the feature/criterion set proposed by our framework is a minimal and consistent superset of the features found in the evaluation frameworks studied.

Further research in this regard can focus on enhancing the quantitative and formal aspects of the proposed framework, and enriching it with proven evaluation techniques. Assessing a select set of methodologies against the framework can improve its reliability and validity. For verifying the proposed framework, it can be used in method chunk selection and end-result evaluation in the context of a realistic method engineering project.

ACKNOWLEDGEMENT

We wish to thank Iran Telecommunications Research Center (ITRC) for sponsoring this research.

REFERENCES

[1] U. Frank, "A comparison of two outstanding methodologies for object-oriented design", Arbeitspapiere der GMD, Nr. 779 Sankt Augustin, 1993 (Online), Available: www.uni-koblenz.de/ ~iwi/publicfiles/PublikationenFrank/RumbaughBooch.pdf [Accessed: October 2009].

[2] International organization for standardization (ISO), International Electro technical Commission (IEC), ISO/IEC: 9126 Software engineering – Product quality; Part 1-4, Geneva, 2004.

[3] K. H. Fung and G. C. Low**, "**Methodology evaluation framework for dynamic evolution in composition-based distributed applications," Journal of Systems and Software, vol. 82, 2009, pp. 1950-1965, doi:10.1016/j.jss.2009.06.032.

[4] B. Kitchenham and L. Jones, "Evaluating software engineering methods and tools. Part 6: identifying and scoring features," ACM SIGSOFT Software Engineering Notes, vol. 22, 1997, pp. 16-18, doi:10.1145/251880.251912.

[5] S. Hong, G. van den Goor, and S. Brinkkemper, "A Formal Approach to the Comparison of Object Oriented Analysis and Design Methodologies," Proc. Twenty-Sixth Hawaii International Conference on System Sciences (HICSS 93), IEEE press, 1993, pp. 689-698, doi:10.1109/HICSS.1993.284253.

[6] Object Management Group (OMG), "Unified Modeling Language Specification (v1.5)", 2003.

[7] R. Ramsin and R. F. Paige, "Process-Centered Review of Object-Oriented Software Development Methodologies," ACM Computing Surveys, vol. 40, 2008, pp. 1-89, doi: 10.1145/1322432.1322435.

[8] P. Cuesta, A. Gomez, J. C. Gonzalez, and F. Rodrıguez, "A Framework for evaluation of agent-oriented methodologies," Proc. Fourth Iberoamerican Workshop on Multi-Agent Systems (Iberagents 02), 2002, pp. 60-65.

[9] Z. O. Akbari and A. Faraahi, "Evaluation Framework for Agent-Oriented Methodologies," Proc. World Academy of Science, Engineering and Technology, 2008.

[10] H. Packard, "An Evaluation of Five Object-Oriented Development Methods," Software Engineering Department, HP Laboritories, Bristol, 1991.

[11] A. Strum and O. Shehory, "A Framework for Evaluating Agent-Oriented Methodologies," Proc. 5th International Bi-Conference Workshop (AOIS 03), Springer, Heidelberg, 2003, pp. 94-109, doi:10.1007/b98189.

[12] M. Taromirad and R. Ramsin, "CEFAM: Comprehensive Evaluation Framework for Agile Methodologies", Proc. 32nd Annual IEEE Software Engineering Workshop (SEW 08), IEEE Press, 2008, pp. 195-204, doi:10.1109/SEW.2008.19.

[13] KH. H. Dam, "Evaluating and comparing agent-oriented software engineering methodologies," MSc. Thesis, RMIT University, Australia, 2003.

[14] R. S. Pressman, Software Engineering: A Practitioner's Approach, 6th ed., McGraw-Hill, 2005.

[15] Ch. Shyam and K. Chris, "A Metrics Suite for Object-Oriented Design," IEEE Transactions on Software Engineering, vol. 20, 1994, pp. 476-492, doi:10.1109/32.295895.

[16] L.H. Rosenberg and L.E. Hyatt, "Software Quality Metrics for Object-Oriented Environments," SATC/NASA Technical Report, SATC-TR-95-1001, 1997.

[17] F. B. Abreu and R. Carapuça, "Candidate Metrics for Object-Oriented Software within a Taxonomy Framework," Journal of Systems and Software, vol. 26, 1994, pp. 87-96, doi: 10.1016/0164-1212(94)90099-X.

[18] A. Schauerhuber, W. Schwinger, E. Kapsammer, W. Retschitzegger, M. Wimmer, and G. Kappel, "A Survey on Aspect-Oriented Modeling Approaches," Technical report, Vienna University of Technology, 2007.

[19] A. Schauerhuber, W. Schwinger, Retschitzegger, and M. Wimmer. "Towards a Common Reference Architecture for Aspect-Oriented Modeling," Proc. 8th International Workshop on AspectOriented Modeling (AOM 06), 2006.

[20] S. Op de beeck, E. Truyen, N. Boucke, F. Sanen, M. Bynens, and W. Joosen, "A Study of Aspect-Oriented Design Approaches," Report CW435, Department of Computer Science, K.U.Leuven, 2006.

[21] E. Figueiredo, A. Garcia, C. S. Anna, U. Kulesza, and C. Lucena, "Assessing Aspect-Oriented Artifacts: Towards a Tool-Supported Quantitative Method," Proc. 9th ECOOP Workshop on Quantitative Approaches in OO Soft. Engineering (QAOOSE 05), 2005, doi:10.1.1.64.2728.

[22] K. Berg, J. M. Conejero, and J. Hernández**,** "Analysis of crosscutting across software development phases based on traceability," Proc. International workshop on Early aspects at ICSE, ACM press, 2006, pp 43-50, doi:10.1145/1137639.1137647.

[23] C. V. Lopes and S. K. Bajracharya, "An Analysis Of Modularity In Aspect Oriented Design," Proc. 4th international conference on Aspect-oriented software development, ACM press, 2005, pp. 15-26, doi:10.1145/1052898.1052900.

[24] S. L. Tsang, S. Clarke, and E. Baniassad, "Object Metrics for Aspect Systems: Limiting Empirical Inference Based on Modularity," Technical report, Trinity College Dublin, 2000.

[25] L. Cernuzzi and G. Rossi, "On The Evaluation Of Agent Oriented Modeling Methods," Proc. Agent Oriented Methodology Workshop, 2002, pp. 21-30.

[26] E. Yu and L. M. Cysneiros, "Agent-Oriented Methodologies - Towards A Challenge Exemplar," Proc. CEUR Workshop, 2002.

[27] J. Sudeikat, L. Braubach, A. Pokahr, and W. Lamersdorf, "Evaluation of Agent–Oriented Software Methodologies – Examination of the Gap Between Modeling and Platform," Proc. 5th International Workshop AOSE 2004, Springer, Heidelberg, 2004, pp. 126-141, doi:10.1007/b105022.

[28] A. Strum and O. Shehory, "Evaluation of modeling techniques for agent-based systems," Proc. 5th International Conference on Autonomous Agents, ACM Press, 2001, pp. 624-631, doi:10.1145/375735.376473.