An Iterative Criteria-Based Approach to Engineering the Requirements of Software Development Methodologies

Raman Ramsin, Richard F. Paige

Department of Computer Science

University of York

Heslington

York, YO10 5DD

United Kingdom

{ramsin@sharif.edu, paige@cs.york.ac.uk}

Phone: +44 1904 433242

Fax: +44 1904 432767

Abstract. Software engineering endeavours are typically based on and governed by the requirements of the target software; requirements identification is therefore an integral part of software development methodologies. Similarly, engineering a software development methodology involves the identification of the requirements of the target methodology. Methodology engineering approaches pay special attention to this issue; however, they make little use of existing methodologies as sources of insight into methodology requirements.

We propose an iterative method for eliciting and specifying the requirements of a software development methodology using existing methodologies as supplementary resources. The method is performed as the analysis phase of a Methodology Engineering process aimed at the ultimate design and implementation of a target methodology. An initial set of requirements is first identified through analyzing the characteristics of the

development situation at hand and/or via delineating the general features desirable in the target methodology. These initial requirements are used as evaluation criteria; refined through iterative application to a select set of relevant methodologies. The finalized criteria highlight the qualities that the target methodology is expected to possess, and are therefore used as a basis for defining the final set of requirements. In an example, we demonstrate how the proposed elicitation process can be used for identifying the requirements of a general object-oriented software development methodology.

Due to its basis in knowledge gained from existing methodologies and practices, the proposed method can help methodology engineers produce a set of requirements that is not only more complete in span, but also more concrete and rigorous.

Keywords: Methodology Engineering, Requirements Engineering, Criteria-Based Evaluation

1 Introduction

A Software Development Methodology (SDM) can be loosely defined as "a recommended collection of phases, procedures, rules, techniques, tools, documentation, management, and training used to develop a system" [1]; but it is easier to grasp when described as consisting of two main parts [2]: a set of modeling conventions comprising a Modeling Language (syntax and semantics); and a Process, which specifies the development activities and their order, provides guidance for monitoring the activities, and specifies what artifacts should be developed using the Modeling Language.
Methodology engineering – or Method Engineering (ME), as it has come to be called – was originally defined as "The engineering discipline to design, construct, and adapt methods, techniques and tools for the development of information systems" [3, 4]. The concept has over the years become mainly restricted to Situational Method Engineering (SME) [5], in which methodologies are constructed or adapted to fit specific project situations. SME is based on the notion that software development processes are software too [6, 7], and applying an engineering approach to their development is therefore logical. It is also motivated by the observation that the one-size-fits-all strategy pursued by older versions of heavyweight methodologies (such as Rational Unified Process – RUP [8] and Object-oriented Process, Environment and Notation – OPEN [9]) is not justifiable, since methodologies typically have to be tailored – engineered – to fit particular project situations. This

observation is now widely accepted in the software engineering community, and as a result, modern methodologies strive to provide configurable and flexible processes that can be adapted to the specific needs of a software development project; examples include heavyweights such as Catalysis and Enterprise Unified Process (EUP), and agile lightweights such as Adaptive Software Development (ASD), Crystal Clear, and Dynamic Systems Development Method (DSDM) [10]. The switch to SME has now resulted in the emergence of methodology assembly frameworks and their corresponding method-chunk repositories, through which custom methodologies can be constructed by assembling reusable method chunks [5]. Most heavyweight methodologies have now been replaced or enhanced by such SME frameworks; for instance, OPEN has become OPEN Process Framework (OPF) [11], RUP has been fused into the Rational Method Composer (RMC) [12], and the Eclipse Process Framework (EPF) has gained widespread recognition as a generally applicable process framework [13].

Eliciting the requirements of a methodology is not a practice exclusive to modern SME approaches. Methodology requirements have long been used for methodology selection and adaptation. The Jackson System Development (JSD) methodology of 1983 is an early example of a methodology that regards characteristics of the problem domain as methodology requirements, and uses them in customizing the methodology and verifying its suitability for the problem at hand [14]. A similar approach can also be observed in some modern adaptive processes [10]. As a further example, the Problem Frames approach [15], and some of the various methods based on it [16, 17], capture methodology requirements as specifications of the problem domain, based on which the software development methodologies that are most appropriate for the project are proposed.

Although methodology requirements have long shown their usefulness, SME is where methodology requirements demonstrate their full significance. In SME projects, methodology requirements may not be restricted to the relevant characteristics of a specific *software-development* problem domain – functional and non-functional requirements of a specific software product, such as quality-related issues, that affect the choice of methodology. Direct

characteristics of the methodology itself, often imposed by the organization, are considered more essential; examples include *Seamlessness* and *Flexibility* of process. Although these characteristics are usually categorized in software engineering as non-functional *development* requirements/constraints [18], in many cases they are common among several projects. It should be noted that methodologies developed through SME may not be targeted at one software development project only; organizations can use SME to develop their own adaptive processes, each aimed at a host of similar projects (or recurring project situations). Therefore, the problem domain of a methodology-engineering project may be broader than that of any of its target software-engineering projects – projects for which the methodology is intended.

As typologies of SME approaches and techniques show [19, 20], and as expected in any software engineering effort, methodology development is heavily dependent on eliciting and specifying the requirements of the target methodology as early in the development process as possible; the methodology development effort will then proceed in a requirements-based fashion. Requirements engineering approaches currently practiced in method engineering rely on the organization or project situation at hand as the sole source of requirements [21, 22, 23]. This seems logical, since the project/organization is in fact the problem domain itself. However, requirements elicited in this fashion are deliberately abstract. Abstract methodology requirements may be achievable at a spectrum of degrees and through diverse methods. While abstractness is desirable when implementation-independence should be observed, it can increase the number of alternative solutions, thereby complicating the design process.

In method engineering approaches currently practiced, existing methodologies are mainly used as sources of method fragments: methodologies are broken down into method chunks, which are then stored in method repositories to later be reused for assembling bespoke methodologies [24]. Ideas from existing methodologies are sometimes captured as guidelines [11], but they are rarely used as resources for method requirements; whereas in software engineering, alternative solutions/architectures and software systems similar to the target system are explored during detailed analysis and early stages of design. Agile methods and third-generation object-oriented methodologies

prescribe the specification of architectural design alternatives even earlier in the process. This means that solution-domain issues are explored prior to design, resulting in enhanced risk mitigation, especially as pertaining to technical and requirements risks. The same approach could also be useful in a methodology engineering context: existing methodologies and techniques that are relevant to the situation at hand can be explored in order to identify their capacities and shortcomings: strengths that can be reused, weaknesses that should be addressed, opportunities for improvement, and pitfalls that ought to be avoided. A better understanding is thereby acquired of the issues involved in developing the target methodology, resulting in a set of requirements that is not only more complete in span, but also more rigorous due to the level of insight gained.

It should be noted that methodology engineering should in no way be considered as strictly limited to Situational Method Engineering, i.e., on-the-fly development of custom methodologies; it can also include the engineering of general-purpose, customizable methodologies. In such cases, existing methodologies can be even more useful as resources, as they not only help enrich the requirements with ideas from the solution domain, but can also lead to new requirements; existing methodologies are thus considered as complements to the SME problem domain. This is somewhat similar to eliciting the requirements of a software system from scratch; that is, when an older version of the system does not exist in the organization.

Motivated by the issues outlined above, we propose an approach to methodology requirements engineering which is based on the notion that a software development methodology can be developed (engineered) via a software engineering process – that is, through the generic development phases of requirements analysis, design, implementation and test. The approach adopted for the requirements analysis phase – reported herein – is based on using existing methodologies and techniques as additional resources when eliciting and specifying methodology requirements. This is somewhat analogous to approaches used for analyzing and designing Domain-Specific Languages (DSL) [25]. We argue that since methodology requirements need to be precise and complete enough to be useful during methodology design, it is worthwhile to enrich them with ideas from the solution domain prior to design. Hence, an iterative criteria-based approach is proposed that, starting from a basic set of

high-level requirements, selects and evaluates the methodologies that are relevant to the project situation at hand. The basic requirements are then used as initial evaluation criteria; they are refined and restructured through iterative application to the selected methodologies, and are ultimately used for specifying the requirements. A concrete specification of the requirements is thus produced that not only defines the desirable level of satisfaction for each requirement, but also provides suggestions for realizing the requirement through using ideas from existing methodologies. The target methodology can then be developed through making use of existing techniques in such a way as to satisfy the requirements.

The rest of this paper is structured as follows: an overview of our proposed iterative criteria-based process and a discussion of its motivations and justifications are presented in Section 2, while section 3 contains a detailed step-by-step description of the elicitation process; Section 4 presents an example that demonstrates how the proposed elicitation process is used for identifying the requirements of a general object-oriented software development methodology; and Section 5 contains the conclusions and several suggestions for furthering this research.

2 The Proposed Iterative Criteria-Based Approach

The merits of criteria-based analysis as a source of insight into the capabilities and shortcomings of software development methodologies has long been recognized, as shown in previous research on software development methodologies in general [26] and object-oriented software development methodologies in particular [27, 28]. The results obtained from such analyses are prevalently used for selecting, tailoring and effective usage of methodologies. Naturally, the main problem that any researcher attempting to exercise such analyses faces is the definition of a suitable set of criteria.

In our proposed approach to methodology requirements elicitation, evaluation criteria are used as intermediate means for refining the requirements: initial high-level requirements are taken as evaluation criteria, which are then refined through iterative application to methodologies, and ultimately converted to detailed and concrete specifications of the requirements. The criteria-refinement method proposed herein is based on the observation that the strengths and weaknesses of

methodologies (identified through criteria-based evaluation) provide further ideas as to what is and what is not desirable in methodologies; this can in turn lead to the identification of new criteria and/or refinements to the existing ones. Thus, the approach uses the evaluation results themselves for refining the criteria. A method can thus be devised to refine an initially incomplete set of evaluation criteria through applying them iteratively to software development methodologies, until the criteria and the evaluation results are stabilized.

The evaluation results appear to be the main output of this process, yet it is the final criterion set that will provide the ultimate objective: a set of requirements for the target software development methodology. This is achieved by evolving each criterion into a requirement through adding the level of support that the target methodology is expected to provide for that criterion, taking into account the lessons learnt from existing methodologies (as inferred from the evaluation results). The details of the requirements elicitation process based on the above approach are explained in the next section.

3 Requirements Elicitation Process

The elicitation process consists of the following stages, during which initial requirements are identified and set as evaluation criteria, the criterion set is refined through iterative criteria-based evaluation of selected methodologies, and the final criteria are turned into requirements (Figure 1). The stages are described through the rest of this section.

Definition of an initial set of requirements

Initial requirements help select the methodologies of interest, and also act as seed criteria for the iterative evaluation stage of the process. The initial set of requirements is extracted from the parameters of the project situation at hand [21, 29, 30], and/or selected from among features that are considered to be desirable in the target methodology [31, 32]. The important point to have in mind is that the initial set of requirements will be applied as the seed criterion-set, and is therefore expected to act as a "detonator": Since the criteria are used as focus-pointers guiding the evaluation process in exposing the processes' strengths and weaknesses, the initial criterion set should be expansive and incisive in order to trigger a large-scale fan-out effect, ever increasing the breadth and depth of the

analysis, and thereby uncovering new criteria and refining the existing ones. It is therefore recommended that certain coverage requirements be added to the initial list to ensure the detonation effect; these are requirements that address the generic-lifecycle coverage of the methodologies (consisting of generic Definition, Development, and Maintenance activities [18, 32]). The initial criterion set is expected to undergo dramatic changes – both in structure and content – during the evaluation process.

Selection of a set of software development methodologies

Relevant methodologies are selected to then be analyzed based on the evaluation criteria. This typically requires studying available resources to gain a better understanding of relevant methodologies. Since the richness of the reviews and the analysis results is of utmost importance when defining the requirements, the set of methodologies should be comprehensive enough to provide extensive coverage of the features targeted; sets of process patterns and process metamodels may also be added to further broaden the coverage. The list of selected methodologies will be revisited and revised during the iterative refinement stage.

Summarization and review of the selected methodologies

The selected methodologies are summarized using a process-centered template, highlighting the activities prescribed in each methodology while keeping the description and discussion of the artifacts produced (and the modeling languages used) as secondary to the activities [10]. The description produced using this template offers little critique on the methodologies, yet abstracts them so that detailed analysis of each individual methodology is facilitated. The description of a methodology based on this template consists of the following parts [10]:

1. An introductory preface providing a brief account of the methodology's history and distinguishing features, as well as an abstract overview of the development process prescribed by the methodology.
2. A number of subsections, one for each high-level subprocess in the methodology's development process, each consisting of:

2.1. Details of the activities performed in the subprocess and the order in which they are performed.

2.2. A concise description of the artifacts produced and the modeling languages used in the subprocess, described as part of their relevant activities.

Iterative refinement process

During the following steps, the evaluation results are incrementally built, and the criterion set – which initially contains the seed criteria – is gradually refined. The final list of criteria is expected to possess a minimum degree of quality, measured through a predetermined set of meta-criteria (criteria for evaluating other criteria) that are iteratively applied to the criteria. The choice of meta-criteria may depend on the problem domain, but it is imperative that the meta-criteria are defined prior to the initiation of the iterative refinement process. The cycle is repeated until the evaluation results and the criterion set are stabilized (i.e., further iterations are unlikely to significantly change the criteria and the evaluation results):

1. Evaluating the selected methodologies based on the criterion set: The significant strengths and weaknesses of the methodologies are thus determined. The criteria are used as focus-pointers, concentrating the evaluation on areas where significant strengths and weaknesses are most likely to be found. Unlike many criteria-based evaluations, the results are not represented as ratings denoting the degree of support each methodology provides for each of the criteria. Furthermore, since the criteria are used as focus-pointers, there is no one-to-one relationship between the criteria and the results: a process might possess several significant strengths/weaknesses as pertaining to one criterion, while having nothing significant to offer in relevance to another criterion. A simple rating procedure would add nothing new to the criteria, whereas the focus-pointing approach makes it possible to gradually increase the span and depth of exploration, and thereby identify potential areas for improving the criteria; the criteria are thus allowed to evolve. Evaluation results may need to be restructured based on the preferences of the analysts: In cases where two or more results are similar, merging may be considered; on the other hand, if an

evaluation result points to two or more distinct strengths/weaknesses that should be stressed in their own right, decomposition may be deemed necessary.

2. Updating (evolving) the criterion set and the list of methodologies: Using the evaluation results as a resource, the following activities are performed:

   2.1. New criteria are added, and refinements are made to existing criteria or their structure. It is perfectly acceptable if there are criteria that cannot be traced to any of the previous (or seed) criteria; this may happen if new criteria are added during iterative refinement. However, if there is a previous (or seed) criterion to which none of the current criteria can be traced, a requirements loss has occurred, which is not acceptable. The important rule is that existing criteria should be maintained, unless they are covered by other criteria. If this rule is not strictly observed, traceability could be compromised. Permissible actions are as follows:

      2.1.1. A new criterion will be added in case a new feature (requirement) is identified for the target methodology. The typical case is when evaluation results point to a new feature; usually, the new feature is either a finer-grained version of an original criterion (e.g., “support for risk estimation” is a finer-grained version of “support for risk management”), or a concrete implementation that satisfies a criterion (e.g., “iterative-incremental nature of the development process” satisfies “support for risk management”). The original criterion is usually kept along with the new one, unless the analysts decide that it is redundant.

      2.1.2. As evaluation proceeds, the definition of each criterion will be refined in order to better express the feature being evaluated.

      2.1.3. Criteria are restructured in order to enhance their understandability. Criteria leading to similar results may be merged, and criteria pointing to different features may be broken down to emphasize the constituent features.

   2.2. The list of methodologies is revisited, and due changes are applied based on the new set of criteria. New methodologies are added if one or more criteria are not adequately addressed by the methodologies that are currently on the list.

Producing and specifying the requirements

The stabilized criterion set is used as a framework for producing the requirements of the target methodology, and also for detailing it using the evaluation results. Since the criterion set can be regarded as a framework that defines the general features desirable in the target methodology, requirements can be specified by detailing and enriching these features with information on the degree of support expected in the target methodology. Consider risk-management as an example of an evaluation criterion: in order to evolve it into a requirement, the degree of risk management support that the target methodology is expected to provide should be defined. Furthermore, development processes offer alternative ways for implementing desirable features. Therefore, evaluation results can be enriched with information as to how criteria are met or contradicted in each of the methodologies evaluated, thus providing a toolkit of methods and techniques for implementing features, as well as a list of potential pitfalls to be avoided. The repertoire of ideas thus built (containing lessons learnt from existing software development processes) can guide the developers in defining and refining the requirements.

4 Example: Eliciting the Requirements of an Object-Oriented Methodology

An Object-Oriented Software Development Methodology (OOSDM) is specifically aimed at viewing, modeling and implementing a system as a collection of interacting objects, using specialized modeling languages, activities and techniques needed to address the specific issues of the object-oriented paradigm. The applicability of the object-oriented approach to systems analysis and design was recognized in the mid 1980s, and the subsequent enthusiasm has been such that a plethora of object-oriented software development methodologies have been introduced [10].

A close look at the present state of affairs in the field of object-oriented software development methodologies shows that despite all the advances made over the years, numerous deficiencies appear in these methodologies, some of which have been summarized in [33, 34]. Nevertheless, OOSDMs are still considered state of the art, and research aimed at improving them is an ongoing process [33,

35]. There is motivation for developing methodologies that use the lessons learnt from the Unified Modeling Language (UML) [36] and the long history of object-oriented methodologies in setting up a framework for software development that addresses the problem issues.

In order to show how the proposed elicitation process works, it has been used for eliciting the requirements of an object-oriented methodology that addresses the deficiencies commonly seen in such methodologies. Object-oriented software development methodologies are a suitable context for such an effort, mainly because of their relatively long history, during which many development problems have been encountered and addressed [10, 31, 35]; the degree of maturity enjoyed by these methodologies means that they are rich resources that can be used for providing the criteria and the requirements.

4.1 Initial Requirements

The following have been observed as core areas where further work on OOSDMs is needed, hence highlighting the general characteristics of an enhanced object-oriented methodology [10]:

1. *Compactness:* an extensible core is preferable to a customizable or generic framework with complex parameters and/or numerous parameter options.
2. *Extensibility:* with extension mechanisms and guidelines clearly defined.
3. *Traceability to requirements:* all the artifacts should be traceable to the requirements.
4. *Consistency:* artifacts produced should not be allowed to contradict each other; alternatively, there should be mechanisms for detecting inconsistencies.
5. *Testability of the artifacts from the start:* this will allow tools to be developed to verify and validate the artifacts.
6. *Tangibility of the artifacts:* artifacts should be concrete enough to be related to and understood by the parties involved in their development or assessment.
7. *Visible rationality:* there should be evident rationality behind every task and the order in which the tasks are performed, and undeniable use for every artifact produced. The developers should be able to see this logic, and have access to information that will help them determine whether digression will put their objectives at risk.

The eight requirements selected as the initial requirements are tabulated in Table 1, to also be used as seed criteria. The seven characteristics listed above have been included in order to reveal what the existing processes lack or provide in this regard, thus unearthing features to exploit and pitfalls to avoid. The first requirement has been added in order to broaden the scope of the evaluation to cover the whole lifecycle of the processes, and also to prompt scrutiny into the details of the activities performed. Each criterion on this list has been designated with an identifier; these identifiers will be used in later sections to demonstrate traceability.

Table 1. Initial Requirements (Seed Criteria)

Requirement (Criterion)	**Explanation**	**Identifier**
Coverage of standard software development activities	Covering activities constituting or supporting the generic software development lifecycle [32].	SC1
Compactness of process	Referring to lightness and simplicity of process, and its being free of nonessential, excess features; hefty and complex processes are hard to understand and master, and difficult to use.	SC2
Extensibility of process	The degree to which the process can be extended to support software development efforts of different sizes, complexities and criticalities.	SC3
Traceability of artifacts to requirements	The degree to which artifacts can be shown to have stemmed from the requirements.	SC4
Consistency of artifacts	Mutual agreement and logical coherence of the artifacts.	SC5
Testability of artifacts	The degree to which artifacts lend themselves to establishment of test criteria and performance of tests to determine whether the test criteria have been met.	SC6
Tangibility and understandability of artifacts to users and developers	The level of consideration given to the balance between abstraction and concreteness in producing the artifacts – removal or reduction of low-level detail when appropriate and developing physical manifestations when necessary (e.g. prototyping) – with the ultimate objective of enhancing the perceptibility of the underlying notions.	SC7
Rationality of process and artifacts	Evident rationality behind every task and the order in which the tasks are performed, and undeniable use for every artifact produced.	SC8

4.2 Selection and Process-Centered Description of Methodologies

Based on the initial requirements of Table 1, a total of 24 object oriented methodologies were chosen as resources, spanning all the three classes of object-oriented methodologies: seminal, integrated and agile [10]. Since a general methodology was targeted, methodologies had to be selected that offered alternative ways for satisfying the initial requirements. This necessitated the selection of a wide spectrum of methodologies. A set of process patterns and process metamodels were also added in order to complement the set of methodologies, thereby enriching the feature set to be used in defining the requirements.

Due to the large number of methodologies, only those most renowned and influential have been selected; methodologies that, according to the evolution timelines of [37] and [38], either have started or are suitable representatives of individual branches of evolution.

It should be noted that object-oriented software development approaches that lack a detailed process or a reasonably-defined process metamodel have not been considered for inclusion. Methodologies based on the Model Driven Architecture (MDA) [39] are the most important of these, as they are not mature enough for our purposes [40].

Table 2 shows the methodologies, process patterns and process metamodels used in this analysis. The methodologies were described using the process-centered template; the reader is referred to [10] for the full descriptions. The list of methodologies is revisited during the iterative refinement process and may be changed according to the current version of the criteria and the evaluation results: for example, when the selected methodologies do not have much to offer as to a specific feature that is evaluated by a recently added criterion. However, in our example, changing the list of methodologies was not necessitated. This was due to the richness of the methodologies originally selected.

Table 2. Object-oriented methodologies, process patterns and process metamodels used in the example

Methodologies	**Seminal**	Shlaer-Mellor (1988, 1992) Coad-Yourdon (1989, 1991) RDD (1990) Booch (1991, 1994)

		OMT (1991) OSA (1992) OOSE (1992) BON (1992, 1995) Hodge-Mock (1992) Syntropy (1994) Fusion (1994)
	Integrated	OPM (1995, 2002) Catalysis (1995, 1998) OPEN (1996) RUP/USDP (1998, 1999, 2000, 2003) EUP (2000, 2005) FOOM (2001, 2007)
	Agile	DSDM (1995, 2003) Scrum (1995, 2001) XP (1996, 2004) ASD (1997, 2000) dX (1998) Crystal (1998, 2004) FDD (1999, 2002)
Process Patterns		Ambler (1998)
Process Metamodels		OPF – as part of the OPEN methodology (2001) SPEM (2002)

4.3 Iterative Criteria-Based Evaluation

As a result of iterative criteria-based evaluation of the selected methodologies and process patterns/metamodels, their significant strengths and weaknesses were identified. As an example, Figure 2 shows the final results obtained for RUP; the complete results for all methodologies can be found in [41]. The identifiers in the brackets are identifiers of the final criteria (Table 3), and show how the results shown in the figure correspond to the final criterion set. It should be noted that the results shown in Figure 2 are the final evaluation results; they are therefore more closely

related to the final criteria of Table 3 than the seed criteria of Table 1. Earlier versions of the results, however, are naturally oriented towards the seed criteria. As iterative refinement proceeds, criteria and results are iteratively revisited and revised. In our example, iterative restructuring and refinement has changed both the criteria and the evaluation results extensively. The evaluation results were used along with the final criteria for defining the requirements of a general OOSDM.

Table 3. Final criterion set

Methodology Component	**Criterion**	**Identifier**
Process	Clarity, rationality, accuracy, and consistency of definition *[SC1,SC8]*	PC1
	Coverage of the generic development lifecycle activities (Analysis, Design, Implementation, Test, Maintenance) *[SC1]*	PC2
	Support for umbrella activities, especially including: *[SC1,SC3]* Risk management Project management Quality assurance	PC3
	Seamlessness and smoothness of transition between phases, stages and activities *[SC4,SC8]*	PC4
	Basis in the requirements (functional and non-functional) *[SC1,SC4,SC8]*	PC5
	Testability and Tangibility of artifacts, and traceability to requirements *[SC4,SC6,SC7]*	PC6
	Encouragement of active user involvement *[SC4,SC7,SC8]*	PV7
	Practicability and practicality *[SC2,SC8]*	PC8
	Manageability of complexity *[SC3,SC8]*	PC9
	Extensibility/Configurability/Flexibility/Scalability *[SC2,SC3,SC8]*	PC10
	Application scope *[SC8]*	PC11
Modeling Language	Support for consistent, accurate and unambiguous object-oriented modeling at different levels of abstraction and diverse degrees of granularity *[SC4,SC5,SC6,SC7,SC8]*	MLC1
	Provision of strategies and techniques for tackling model inconsistency and managing model complexity *[SC5,SC7,SC8]*	MLC2

4.4 Final Criterion Set

The final, stabilized version of the criterion set, refined as the result of iterative application to the selected methodologies and process patterns/metamodels, is shown in Table 3. The identifiers in the brackets show how each final criterion can be traced to the seed criteria of Table 1.

The validity meta-criteria of [26] – which are specifically intended for appraising criteria that are used for evaluating object-oriented methodologies – were used in the example as quality meta-criteria. The final criteria satisfy these meta-criteria in that they are:

1. general enough to be used for evaluating all object-oriented software development methodologies,
2. precise enough to help discern and highlight the similarities and differences among object-oriented software development methodologies,
3. comprehensive enough to cover all significant features of object-oriented software development methodologies, and
4. balanced: adequate attention has been given to all three major types of features in a methodology: technical, managerial and usage [26].

A closer look at the final criteria shows the effects of iterative refinement. To get a better understanding of the effect of the refinement process, consider the first seed criterion (SC1: Coverage of standard software development activities) as an example. As refinement progressed, it was observed that evaluation results showed two aspects of coverage: coverage of generic software development activities (ultimately evolved into PC2), and inclusion of management (umbrella) activities (ultimately evolved into PC3). It was also observed that an important factor in assessing coverage was the availability of proper documentation (later evolved into PC1). It should be noted that in many cases, evaluations based on two separate criteria gave similar or related results; in such cases, the evaluation results were merged, sometimes even resulting in a merger of their corresponding criteria.

Compared to the seed criteria, the final criterion set is more complete, finer-grained, and better structured. The restructuring that has been applied to the criteria during iterative refinement has resulted in a many-to-many relationship between the seed criteria and the final criteria.

4.5 Final Requirements

Using the final criterion set as the basis, and applying the lessons learnt from the results of the criteria-based evaluation of object-oriented software processes, the following requirements have been identified for the target object-oriented software development methodology:

PR - Process Requirements

PR 1- Definition: The methodology should be well-documented (a comprehensive, clear, rational, accurate, detailed and consistent description should be provided):

1. What should be captured? Lifecycle and work-units, producers, modeling language, work-products, techniques and rules, and issues pertaining to umbrella activities. Metamodels suggested by SPEM and OPF provide useful information as to what should be captured in the definition.
2. How? Mainly process-centered: the structure of the documentation should closely resemble that of the lifecycle, and everything should be described as secondary to the work-units (phases, stages and activities) of the lifecycle. Gradual refinement (hierarchical layering) should be used in describing the process. Role-centered and product-centered views of the methodology can also be added as complements, as suggested by SPEM and OPF – which regard processes as consisting of work-units, roles (producers), and products (artifacts). The role-centered view focuses on the producers involved in the methodology, describing the work-units they participate in and the artifacts they produce, while the product-centered view focuses on the artifacts produced, describing the work-units where they are produced and the producers (roles) involved.

PR 2- Coverage: The generic software development lifecycle activities (Definition, Development, and Maintenance) should be covered. Fusion, RUP, EUP and Catalysis are examples of methodologies providing extensive coverage. Close examination of the generic software development lifecycle [18, 32], Ambler process patterns [42], and the OPEN Process Framework (OPF) [11] shows that the following activities should be covered as a minimum:

1. Definition
   1.1. Problem domain exploration and modeling
   1.2. Requirements elicitation

1.3. Feasibility analysis

2. Development

   2.1. Architectural Design

   2.2. Detailed Design

   2.3. Programming

   2.4. Test

   2.5. Deployment

3. Maintenance

PR 3- Support for umbrella activities: Especially including:

1. Risk management: through risk assessment and risk mitigation activities incorporated into the lifecycle. Of special importance are techniques proven effective in other methodologies: e.g. preliminary feasibility analysis (as seen in Crystal Clear and DSDM), prototyping (e.g. RUP, DSDM, and XP), risk-based planning (e.g. RUP, DSDM and Scrum), iterative-incremental development (e.g. RUP and agile methodologies), active user involvement (e.g. Scrum and FDD), continuous verification and validation (e.g. Hodge-Mock, XP and ASD), iterative process/product/plan reviews (e.g. ASD and Crystal Clear), early releases (e.g. XP and DSDM), and continuous integration (e.g. XP and Crystal Clear).
2. Project management: through planning, scheduling and control techniques incorporated into the process (as in RUP and EUP; DSDM and Scrum are good agile examples). Provision should be made for the plans and schedules to be iteratively revisited and revised based on experience gained through the development (as in EUP, ASD and Scrum). Special attention should be given to team management aimed at enhancing intra-team and inter-team communication and collaboration (as seen in RUP, EUP, Scrum and ASD).
3. Quality assurance: through quality assessment and enhancement techniques incorporated into the process. Of special importance are techniques proven effective in other methodologies: e.g. iterative technical reviews (agile methodologies; e.g. ASD and Crystal Clear), design by contract (e.g. BON), continuous verification and validation (e.g. Hodge-Mock, XP and DSDM), and strategies/techniques enhancing requirements traceability (e.g. use-case-driven methodologies

such as OOSE and RUP, scenario-based methodologies such as Hodge-Mock, and agile methodologies such as DSDM and FDD).

PR 4- Seamlessness and smoothness of transition between phases, stages and activities: Although seamlessness can be incorporated via basing all tasks and artifacts on a common concept (e.g. classes in BON, the Domain Model in Shlaer-Mellor, and use cases in RUP), the transition between phases, stages and activities is not necessarily smooth, since it might involve the production of brand new artifacts; even though not violating seamlessness, the effort that is typically required damages smoothness of transition. An alternative seamless strategy is continuous refinement of a specific set of models, around which the development tasks are oriented, which provides both seamlessness and smoothness of transition (as used in Coad-Yourdon, Syntropy and Catalysis). Fractal modeling (as in Catalysis) is an example of a technique that is particularly successful in this context. It should be noted that all methodologies providing smooth transition are not necessarily seamless; many agile methodologies provide smooth transition because of the iterative-incremental nature of their development strategy and the short cycles they usually have, yet they cannot always be considered seamless, since there can be a huge gap between analysis and implementation.

PR 5- Basis in the requirements (functional and non-functional): Functional and non-functional requirements should be captured early in the process, modeled in their own right, and used as a basis for design and implementation (Coad-Yourdon is an example of a methodology that neglects this seemingly obvious requirement). Requirements-driven methodologies – such as use-case-driven methodologies (e.g., Catalysis and RUP), and agile methodologies (e.g., FDD and XP) – are good examples of successful methodologies in this regard. Requirements should be allowed to evolve during the process, as is the case in agile methodologies.

PR 6- Testability and tangibility of artifacts, and traceability to requirements: Artifacts should be few, simple, and understandable, with dependencies that are minimal and clearly defined (Catalysis is a good example, as are many seminal methodologies, e.g. Fusion). Artifacts should complement each other in the context of the process, not decorate each other with clutter. Tangibility of the artifacts to the users and the developers should be maximized: executable artifacts and artifacts with syntax and semantics understandable to the user are tangible to the user (as in DSDM), while

developers find those artifacts tangible that are visibly useful in the process (otherwise they will be ignored or botched, and quality may suffer as a result). Artifacts should be traceable to the requirements (e.g., as direct or indirect realizations of the requirements – as in RUP, FDD, and XP, or via the use of requirements-based evaluation scenarios – as in Hodge-Mock).

PR 7- Encouragement of active user involvement: This is vital for risk management and quality assurance. Ambassador users, and planning and review sessions with user participants are proven techniques. Agile methodologies have a great deal to offer in this regard.

PR 8- Practicability and practicality: The methodology should be employable; and effectively, efficiently and usefully at that. Over-complex methodologies are not practicable; configurability does not solve the problem since it typically involves complex procedures (as is the case with RUP), and neither do instantiation frameworks (like OPEN), for the same reason. Practicability can also depend on the project in hand; performing a feasibility analysis task early in the process (possibly involving the deployment of suitability filters, as in DSDM) is essential. There are numerous factors, other than complexity, that affect practicality (some adversely), and should therefore be taken into account. Tasks that distract the developers from mainstream activities or encumber them with impertinent or unnecessary details should be deleted; techniques and strategies for focusing the development, such as requirements-based models (such as those seen in Fusion, Catalysis and FDD), system architecture/metaphor (such as those seen in RUP and XP), and team management sessions (such as those seen in Scrum and Crystal Clear), have been proven successful in this context. Dependence on error-prone techniques and strategies can damage practicality (such is the over-dependence of some agile methods on the efficacy of human communication, and dogged adherence of some integrated methodologies to UML). Dependency on special tools and technologies can also be detrimental to practicality. A very important factor affecting practicality is the project management strategy; lack of adequate management measures can render the methodology impractical or even impracticable, especially in large projects with stringent constraints on time and resources.

PR 9- Manageability of complexity: The complexity of work-units should be manageable, e.g. via partitioning and layering. Catalysis is a particularly successful example.

PR 10- Extensibility/Configurability/Scalability/Flexibility: The process should be an extensible core, with extension points and mechanisms explicitly specified. It is desirable to be able to configure the extensions or even the core itself in order to fit it to the project at hand (process patterns can be useful in this context). The methodology should be applicable to projects of different sizes and criticalities (as seen in integrated methodologies such as RUP and Catalysis, as well as some agile ones such as FDD and the Crystal family). It should also be dynamically flexible: it should be possible to tune the methodology according to the experience gained during the development; useful techniques are iterative process review sessions, and feedback-based revisions (as seen in ASD and Crystal Clear); it should be noted, however, that tuning is a project-wide decision, and individual teams and developers should not be allowed to make alterations that may have broad implications.

PR 11- Application scope: The application scope depends on the intended usage context, yet targeting information systems as a general usage context seems to be a logical minimum requirement, as this is likely to address the minimum modeling needs of a general methodology (Catalysis and Fusion are prominent examples).

MLR - Modeling Language Requirements

MLR 1- Support for consistent, accurate and unambiguous object-oriented modeling: Specifically covering:

1. Diverse modeling viewpoints: Structural – Functional – Behavioral (as seen in UML, and the modeling languages of OMT and OSA)
2. Logical to Physical modeling: Business-Process/Problem Domain to Solution Domain to Implementation Domain (as seen in UML and OPEN/OML)
3. Diverse levels of abstraction and granularity: Enterprise level – System level – Subsystem/Package level – Inter-object level – Intra-object level (as seen in UML, and the modeling languages of Hodge-Mock and Fusion)
4. Formal and Non-formal specifications (as seen in UML/OCL, and the modeling languages of BON and Syntropy)

Although UML is rich and extensible enough to provide ample support, strict adherence to UML should not be enforced. The use of data-flow diagramming for functional problem-domain modeling is an example of complementing UML with other modeling languages; examples include EUP (where DFDs are used for business modeling) and FOOM (where object-oriented extensions of DFDs are extensively used).

MLR 2- Provision of strategies and techniques for tackling model inconsistency and managing model complexity: Tackling model inconsistency is usually up to the process component of the methodology rather than the modeling language; yet modeling languages can facilitate consistency-checking through providing semantics which define model dependencies and constraints. UML lacks such semantics [39], leaving it to the methodology process to define them; Catalysis is an example of a successful process in this regard. However, modeling languages proposed by many seminal methodologies offer such semantics (examples include BON and Fusion). Another noteworthy contribution in this regard is OPM's single-model approach, which facilitates consistency-checking through eliminating model multiplicity. Modeling languages should also include constructs facilitating complexity management; UML package and component elements are apt examples.

5 Conclusions

The methodology requirements engineering approach introduced in this paper elicits and specifies the requirements of a target software development methodology by using existing methodologies as supplementary resources. It therefore builds on concrete features that are already present in existing methodologies; as demonstrated through the example, this enables the analysts to provide concrete and tangible specifications for the requirements.

The criterion set is initially set to a list of high-level requirements, which are identified through conventional situational method engineering approaches (i.e., based on project parameters) and/or by defining the general characteristics that should be present in the target methodology. This ensures that the process builds on methodology requirements that have been extracted from the problem context.

The example shows the effectiveness of this feature: seed criteria (initial requirements) are covered by (i.e., addressed by) the final criteria as well.

The proposed criteria-based evaluation approach is based on iterative review of selected methodology processes, thereby incrementally identifying the strengths and weaknesses of the processes, refining the set of criteria along the way. The iterative nature of the refinement process ensures that the criteria (requirements) can be constantly reviewed and polished to the desirable degree of completeness and precision. The example presented herein seems to confirm this, as the final criterion set is by far more complete, finer-grained and better structured than the initial seed criteria.

The products of the evaluation process are the evaluation results (lists of strengths and weaknesses for the processes), and the refined criterion set. The requirements are produced through specifying the degree of support expected to be provided by the target methodology for each criterion in the final criterion set. The list of strengths and weaknesses is also used in specifying the requirements: the strengths and weaknesses identified in existing processes show how existing processes meet (or fail to meet) the requirements. This means that they can be used for providing a more detailed specification of the requirements through proposing useful techniques (and warning of potential pitfalls) encountered in existing methodologies. The example shows the potential benefits of this feature, as the final requirements consider practical issues as well, and are therefore more rational in definition.

We have applied our proposed method to the problem of eliciting and specifying the requirements of a general object-oriented methodology. Future research can be focused on applying the approach to elicit and specify requirements for general methodologies belonging to other paradigms, or even specific application domains. However, the applicability of the approach in this context depends on the maturity of the problem domain to which it is applied, since it requires the existence of a set of established methodologies from which to extract the final set of evaluation criteria (and hence, the requirements). Agent-oriented development, for example, is a suitable candidate for furthering this research in this direction. There is also potential for enriching methodologies in poorly-defined problem domains by using features from more mature domains. This requires a multi-domain

analysis, which is expected to be more complex, but it is an opportunity that is definitely worth exploring.

Another direction for furthering this research is to integrate the proposed methodology *analysis* process with a matching methodology *design* process. An instance of such a design process, called the *Hybrid* methodology design method, has been developed by the authors as a complement to the requirements engineering process that has been proposed herein. Hybrid uses an iterative process to systematically apply alternative methodology design techniques to produce a blueprint of the target methodology. This process has been used in designing a new general object-oriented methodology [41], and an agile methodology for developing mobile software systems [43].

The main application context for the proposed method is situational method engineering. Future research can proceed to test the requirements engineering method (and its corresponding design process) in the context of an industrial SME project. An alternative strand of research can focus on integrating the proposed methodology requirements engineering method with existing assembly-based SME approaches.

References

[1] Avison, D.E., and Fitzgerald, G.: 'Information Systems Development: Methodologies, Techniques and Tools' (McGraw-Hill, 2003, 3rd edn.)

[2] Object Management Group (OMG): 'Unified Modeling Language Specification (v1.5)' (OMG, 2003)

[3] Brinkkemper, S.: 'Method engineering: engineering of information systems development methods and tools', Information and Software Technology, 1996, 38, (4), pp. 275-280

[4] Kumar, K., and Welke, R.J.: 'Method engineering: a proposal for situation-specific methodology construction', in Cotterman, W.W., and Senn, J.A. (Eds.): 'Systems Analysis and Design: A Research Agenda' (Wiley, 1992), pp. 257-268

[5] Ralyté, J., Brinkkemper, S., and Henderson-Sellers, B. (Eds.): 'Situational Method Engineering: Fundamentals and Experiences' (Springer, 2007)

[6] Osterweil, L.J.: 'Software processes are software too'. Proc. Int. Conf. Software Engineering, Monterey Canada, March 1987, pp. 2-13

[7] Osterweil, L.J.: 'Software processes are software too, revisited: An invited talk on the most influential paper of ICSE 9'. Proc. Int. Conf. Software Engineering, Boston, USA, May 1997, pp. 540-548

[8] Kruchten, P.: 'Rational Unified Process: An Introduction' (Addison-Wesley, 2003, 3rd edn.)

[9] Graham, I., Henderson-Sellers, B., and Younessi, H.: 'The OPEN Process Specification' (ACM Press/Addison-Wesley, 1997)

[10] Ramsin, R., and Paige, R.F.: 'Process-centered review of object-oriented software development methodologies', ACM Computing Surveys, 40, (1), 2008, Article 3, pp. 1-89

[11] Firesmith, D., and Henderson-Sellers, B.: 'The OPEN Process Framework: An Introduction' (Addison-Wesley, 2001)

[12] Kroll, P.: 'Introducing IBM Rational Method Composer', The Rational Edge, January 2005

[13] Haumer, P.: 'Eclipse Process Framework Composer – Part 1: Key Concepts' (EPF Project, 2006)

[14] Jackson, M., 'System Development' (Prentice-Hall, 1983)

[15] Jackson, M., 'Problem Frames' (Addison-Wesley, 2001)

[16] Kovitz, B., 'Practical Software Requirements: A Manual of Content and Style' (Manning Publications, 1999)

[17] Bray, I., 'An Introduction to Requirements Engineering' (Addison-Wesley, 2002)

[18] Sommerville, I.: 'Software Engineering' (Addison-Wesley, 2004, 7th edn.)

[19] Ralyté, J., Deneckére, R., and Rolland, C.: 'Towards a generic model for situational method engineering'. Proc. Conf. Advanced Information Systems Engineering, Klagenfurt, Austria, June 2003 , pp. 95-110

[20] Ralyté, J., Rolland, C., and Deneckére, R.: 'Towards a meta-tool for change-centric method-engineering: A typology of generic operators'. Proc. Conf. Advanced Information Systems Engineering, Riga, Latvia, June 2004 , pp. 202-218

[21] Gupta, D., and Prakash, N.: 'Engineering methods from method requirements specifications', Requirements Engineering, 2001, 6, (3), pp. 135-160

[22] Ralyté, J.: 'Requirements definition for the situational method engineering'. Proc. IFIP WG8.1 Working Conf. Engineering Information Systems in the Internet Context, Kanazawa, Japan, September 2002, pp. 127-152

[23] Leppänen, M.: 'Conceptual analysis of current ME artifacts in terms of coverage: A contextual approach'. Proc. Int. Workshop on Situational Requirements Engineering Processes, Paris, France. August 2005, pp. 75-90

[24] Mirbel, I., and Ralyté, J.: 'Situational method engineering: combining assembly-based and roadmap-driven approaches', Requirements Engineering, 11, (1), 2006, pp. 58-78

[25] van Deursen, A., Klint, P., and Visser, J.: 'Domain-specific languages: an annotated bibliography', SIGPLAN Notices, 35, (6), 2000, pp. 26-36

[26] Karam, G.M., and Casselman, R.S.: 'A cataloging framework for software development methods', IEEE Computer, 26, (2), 1993, pp. 34-45

[27] Abrahamsson, P., Warsta, J., Siponen, M.T., and Ronkainen, J.: 'New directions on agile methods: A comparative analysis'. Proc. Int. Conf. Software Engineering, Portland, USA, May 2003, pp. 244-254

[28] Monarchi, D.E., and Puhr, G.I.: 'A research typology for object-oriented analysis and design', Commun. ACM, 35, (9), 1992, pp. 35-47

[29] Rolland, C., and Prakash, N.: 'A proposal for context-specific method engineering', in Brinkkemper, S., Lyttinen, K., and Welke, R.J. (eds.): 'Method Engineering Principles of Method Construction and Tool Support' (Chapman & Hall, 1996), pp. 191–208

[30] Slooten, K., and Hodes, B.: 'Characterising IS development projects', in Brinkkemper, S., Lyttinen, K., and Welke, R.J. (eds.): 'Method Engineering Principles of Method Construction and Tool Support' (Chapman & Hall, 1996), pp. 29–45

[31] Graham, I.: 'Object-oriented Methods: Principles and Practice' (Addison-Wesley, 2001, 3rd edn.)

[32] Pressman, R.S.: 'Software Engineering: A Practitioner's Approach' (McGraw-Hill, 2004, 6th edn.)

[33] Boehm, B., and Turner, R.: 'Balancing Agility and Discipline: A Guide for the Perplexed' (Addison-Wesley, 2004)

[34] Boehm, B., and Turner, R.: 'Management challenges to implementing agile processes in traditional development organizations', IEEE Software, 22, (5), 2005, pp. 30-39

[35] Capretz, L.F.: 'A brief history of the object-oriented approach', ACM SIGSOFT Software Engineering Notes, 28, (2), 2003

[36] Object Management Group (OMG): 'Unified Modeling Language Specifications (v2.1.2)' (OMG, 2007)

[37] Webster, S.: 'On the evolution of OO methods' (Bournemouth University, 1996)

[38] Abrahamsson, P., Salo, O., Ronkainen, J., and Warsta, J.: 'Agile Software Development Methods: Review and Analysis' (VTT Publications, 2002)

[39] Object Management Group (OMG): 'Model Driven Architecture (MDA)' (OMG, 2001)

[40] Asadi, M., and Ramsin, R.: 'MDA-based methodologies: An analytical survey'. Proc. Euro. Conf. on Model Driven Architecture Foundations and Applications, Berlin, Germany, June 2008, pp. 419-431

[41] Ramsin, R.: 'The Engineering of an Object-Oriented Software Development Methodology', PhD Thesis, University of York, 2006, http://www.cs.york.ac.uk/ftpdir/reports/YCST-2006-12.pdf

[42] Ambler, S.W.: 'Process Patterns: Building Large-Scale Systems Using Object Technology' (Cambridge University Press, 1998)

[43] Rahimian, V., and Ramsin, R.: 'Designing an agile methodology for mobile software development: A hybrid method engineering approach'. Proc. IEEE Int. Conf. Research Challenges in Information Science, Marrakech, Morocco, June 2008, pp. 351-356

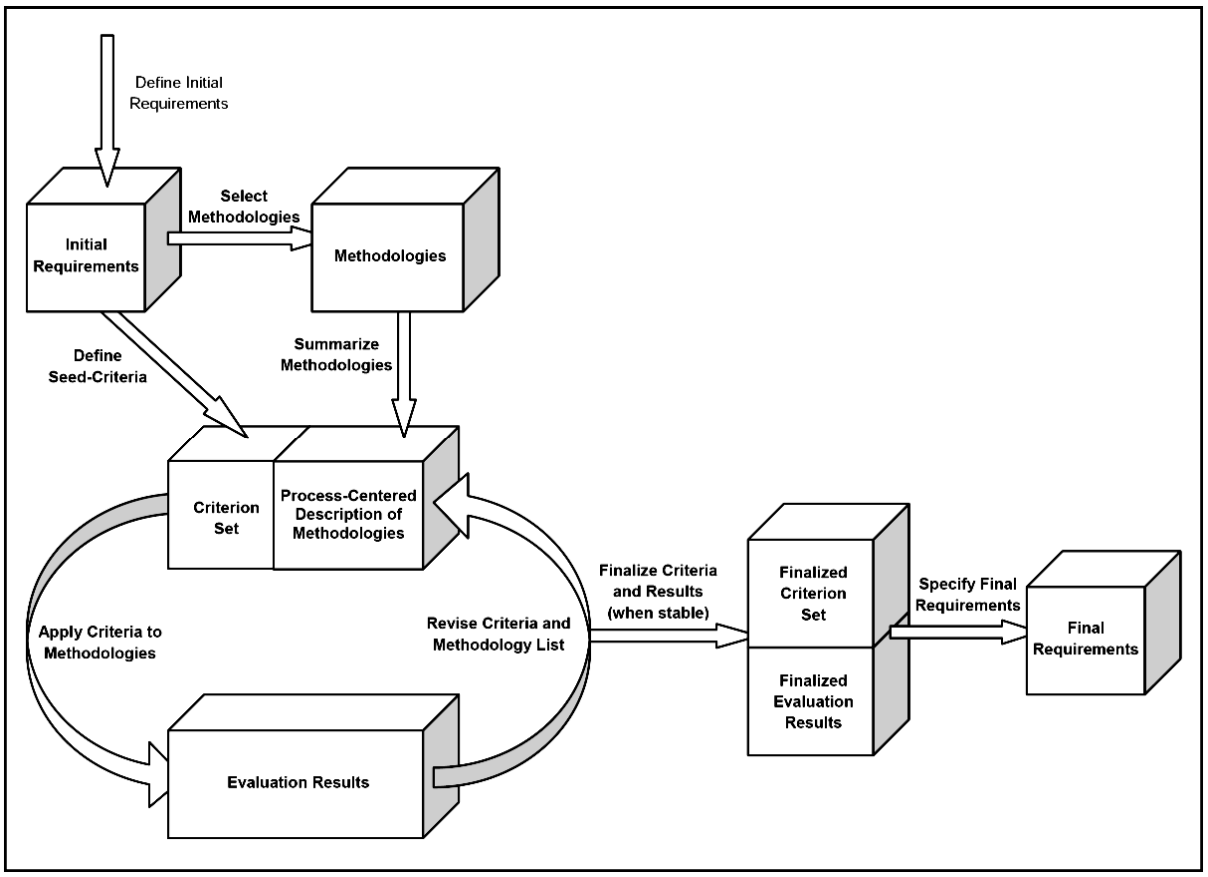

Figure 1. The proposed methodology requirements elicitation and specification process

Strengths

- ☑ Well-documented process *[PC1]*
- ☑ Based on functional, behavioural, and structural modeling of the problem domain and the system *[PC2,PC5]*
- ☑ Process support for structural, behavioural and functional modeling at all levels (problem domain to objects; logical to physical) *[PC2,PC6,PC9]*
- ☑ Iterative-incremental process *[PC3]*
- ☑ Risk-based development, aimed at mitigating the risks before undertaking the tasks *[PC3]*
- ☑ Seamlessness (though with hiccups, e.g. transforming use cases to sequence diagrams) *[PC4]*
- ☑ Traceability supported through use cases *[PC6]*
- ☑ Customizability addressed *[PC8,PC10,PC11]*
- ☑ Architecture-centric process (which necessitates early specification of an architectural blueprint) *[PC9]*
- ☑ Rich modeling language (UML), especially in structural and behavioural modeling features *[MLC1,MLC2]*
- ☑ Support for formalism (through UML/OCL) *[MLC1]*

Weaknesses

- ☒ Very complex process *[PC1,PC8]*
- ☒ Since the process is very complex, not having a maintenance phase, on the grounds that it can be performed by iterating the whole process as a cycle, is not convincing. *[PC2]*
- ☒ The process is confusing to those involved: it is hard to understand the logic behind some of the deliverables and tasks performed. The iterative-incremental nature of the process further complicates the issue. *[PC8]*
- ☒ Inadequate attention to continuous user involvement *[PC7]*
- ☒ Although advertised as customizable, configuring the process is a formidable task in itself. Trying to tailor down the process often has the opposite effect. *[PC10]*
- ☒ Prohibitive number of models *[MLC2]*
- ☒ Strict adherence to UML, which is not necessarily constructive, especially since UML is not perfect and can exacerbate the model inconsistency problem. *[MLC2]*

Figure 2. Example of evaluation results: strengths and weaknesses identified in RUP