IMPORTANT:
This is a draft submitted to OMG for review. It is subject to further changes before actual issuing the public. The purpose of this releasing is to have an update on Semat progress.

Object Management Group

140 Kendrick Street
Building A Suite 300
Needham, MA 02494
USA

Telephone: +1-781-444-0404
Facsimile: +1-781-444-0320

A Foundation for the Agile Creation and Enactment of Software Engineering Methods

Request For Proposal

OMG Document: ad/2011-05-0?

Letters of Intent due: November 22, 2011
Submissions due: February 22, 2012

Objective of this RFP

An agile approach to software engineering methods is one that supports practitioners of software engineering (architects, designers, developers, programmers, testers,

deployers, analysts and project managers) in dynamically adapting and customizing their methods during the preparation and execution of a project, controlled through company specific governance, use of examples and other means. In contrast to previous approaches in this area, which have provided support mainly for process engineers, the goal here is to provide direct support for practitioners as the main target group.

The objective of this RFP is to obtain a foundation for the agile creation and enactment of software engineering methods by development practitioners themselves. This foundation is to consist of a *kernel* of software engineering domain concepts and relationships that is extensible (scalable), flexible and easy to use, and a domain-specific modeling *language* that allows developers to describe the essentials of their current and future practices and methods. These practices and methods can then be supported by tools based on this common foundation, and they can further be composed, simulated, applied, compared, enacted, tailored, used, adapted, evaluated and measured by practitioners as well as taught and researched by academic and research communities.

For further details see Chapter 6 of this document.

1.0 Introduction

2.0 Architectural Context

3.0 Adoption Process

4.0 Instructions for Submitters

5.0 General Requirements on Proposals

6.0 Specific Requirements on Proposals

6.1 Problem Statement

It is widely recognized that the successful development of significant software systems benefits from the application of effective methods and well-defined processes. However, even after more than forty years of work in software engineering, methods are still too often applied haphazardly by development teams.

The traditional prescription for resolving such problems is to carefully define the methods to be used in a software development effort and then provide the development team with the tools to enact the methods so defined. But such software method engineering approaches are often considered by development teams as being too heavyweight and inflexible. Software developers today are used to agility and flexibility in carrying out their projects. However, software methods defined by separate method engineers typically do not leave enough flexibility for a development team themselves, without a method engineering specialist, to customize, tailor and adapt the process they use, not just at the beginning, but *continuously* as necessary over the course of a development effort.

A similar concern has sometimes been raised against model-driven approaches in general, based on the perception that they require detailed modeling to be done "all up front", often by modelers who are not developers. On the other hand, many development teams do use UML or other less formal notations all the time to sketch out designs *amongst the developers themselves.* And the most effective model-driven efforts integrate "agile modeling" by regular developers as a normal practice within their development effort rather than as a heavyweight "additional thing".

Similarly, current software method and process modeling approaches are perceived to require methods to be modeled "all up front", by method engineers who are not developers. This disconnects "method engineering" from the day-to-day work of development practitioners and requires an investment in additional staff and infrastructure that can be hard to justify except in other than large companies working on large projects. Projects come in all different sizes and there is a huge amount of software development that occurs in small to mid-sized companies that do not and realistically will not have method engineers at all.

On the other hand, developers pragmatically talk all the time about what is and is not working in their development method and how to adjust it. But, not only has there been far less acceptance of any consistent notation/language to use in this discussion than, say, UML is used for modeling the system itself, developers have no good codification of the software engineering practices on which effective methods should be based.

As a result, the method discussions within development teams often end up being either overly influenced by the visibility of the latest "hot" approaches or are limited to

tailoring around the edges of some method that has been dictated to them. This seriously limits the ability of a team to be effective and scalable while remaining flexible and agile.

6.2 Scope of Proposals Sought

The use of an application framework of some sort is generally recognized these days as an effective way to enhance the productivity of a development team, particularly in conjunction with an agile development process. Such frameworks provide both a toolkit of components and an easy-to-use scripting language for flexibly composing the components.

In order to address the issues discussed in Section 6.1, development teams should similarly have available to them a framework that allows for the rapid construction of software methods for the team's own use. Such a framework would then allow the team to agilely tailor their methods as they see fit during the course of a development effort. The crucial intent is to support the needs of the software development practitioners themselves, as they see them.

Rather than standardizing on one such framework, though, it would be more desirable to allow industry to develop various frameworks that may be useful in various different contexts. But, to avoid "silos" of non-interoperable frameworks (which is one downside of the current plethora of frameworks in the area of application development), and to promote the adoption of sound software engineering best practices, all these frameworks should be based on a common, standardized foundation. That is what is being requested in response to this RFP.

6.2.1 Methods and Practices

A *method* may be defined as a systematic way of doing things in a particular discipline. For the purpose of this RFP, the relevant discipline is software engineering. Software engineering methods support tasks such as the development of a new software system, the maintenance of an existing system or even the integration of an entire enterprise system architecture.

(Note that the terms "method" and "process" are often used interchangeably in discussions of software engineering. However, there is actually no clear consensus in the community on whether these are really the same thing, or whether methods contain processes or processes contain methods. For the present discussion, these two terms can be considered to be largely synonymous, though in the remainder of the RFP the term "method", in the sense that it is defined in this section, will be consistently used in preference to "process".)

Methods at this level may be considered as composed from well-defined *practices.* A practice is a general, repeatable approach to doing something with a specific purpose in mind, providing a systematic and verifiable way of addressing a particular aspect of the

work at hand. It should have a clear goal expressed in terms of the results its use will achieve and provide guidance on what is to be done to achieve the goal and to verify that it has been achieved. Such practices may include specific approaches for software design, coding, testing at various levels, integration, organizing and managing the development team, etc. Examples of practices are Iterative Development, Use Case Driven Development and Test Driven Development.

Note that the definition of a practice is intentionally similar to that of a method. Indeed, practices at various levels may be composed from lower-level practices, and a method may be considered to be simply a composite practice targeted at the level of support of an entire discipline. This also allows for the further composition of methods at even higher levels within and across disciplines.

6.2.1.1 Enactment of Methods

A further distinction, however, is that a method must be *enactable,* while a practice in isolation will in general not be. In the context of this RFP, the enactment of a method can be defined as the carrying out of that method in the context of a specific project effort. Within this context, the practices within the method may be considered use cases for the work the must be carried out to achieve the project objectives, with each practice providing a specific aspect of the overall method.

Enacting a method includes using the method to create elements such as tasks and work products during the software endeavor, focusing on *what* to produce and *how* to produce it. Since software development is a collaborative team-based effort that involves knowledge workers, enactment of methods should support monitoring and progressing the software endeavor through human agents foremost and automation secondly.

Enactment is always done by practitioners and involves that members of a team collaborates on decision-making, planning and execution. Enactment may be partially supported by method repositories and process engines that are linked to tools such as project management and issue tracking systems.

6.2.1.2 Composition of Practices

The composition of a method from practices is more than just a juxtaposition or grouping of a chosen set of practices. While each practice must be individually definable, a practice will of necessity have expectations on work done outside its scope that must be met by other practices within a complete method. For example, a testing practice will place certain expectations on the concept of a *unit* that is central to a coding practice and the concept of a *testable requirement* generated by an analysis practice. A good method framework must provide effective, easy-to-use means for resolving the expectations across a consistent set of practices and weaving them together into a complete, enactable method.

6.2.1.3 Practice Infrastructure

In this light, this RFP requests the foundation for a common "practice infrastructure" that would enable software developers to more quickly understand, compose and compare individual practices and entire methods. It could also form the basis for the appropriate governance of software organizations, while allowing their developers the freedom to use their preferred practices, composed with those of their organizations. Further, it would allow the evaluation and validation of comparable method and process elements, guide practical research to useful results and act as a common context for training and education.

6.2.2 The Kernel

In order to allow the broadest possible applicability, what needs to be standardized are not practices themselves, but a common *kernel* of underlying concepts and principles applicable across all methods that may be used to define various practices. Such a kernel may be specified as a *domain model* for software engineering, providing a common terminology of concepts and their relationships that may be used in the definition of practices. Kernel elements will include concepts of things to produce or have, such as system, requirement, team, etc. as well as concepts for things to do, such as activity, practice and method.

Further, the concepts in the kernel must be defined in a way that allows them to be *extensible* and *tailorable* to support the needs of a wide variety of practices and methods and to allow flexibility in the definition and application of those methods by each development team. Such extensibility may be provided through a specialization mechanism and/or "slots" or "templates" that are expected to be filled in when concepts are instantiated. However, it is of particular interest for responses to this RFP to provide innovative proposals for extensibility of the kernel.

The key is to provide a powerful enough extensibility mechanism that the kernel itself can stay focused and light, while still allowing it to be easily and naturally applied. This is what distinguishes what is being requested as a *kernel,* as opposed to a new "unified" method framework. The kernel should reflect the *essence* of software engineering in a way that can be widely accepted, and which does not change as new practices are built on it.

6.2.3 The Language

In addition to the kernel, the foundation requested by this RFP includes a standard *language* for specifying practices based on the kernel and for composing methods from the practices. To support a model-driven approach to method engineering, the language should be a *modeling* language that can be used by a development team to both informally discuss and sketch their methods and then formalize those methods as they find appropriate.

As mentioned in Section 6.2.1, enactment is considered here to be the carrying out of a method in the context of a specific project effort. In a certain sense, the requested

language will essentially be used to "script" methods for enactment, in an analogous way to how scripting languages are used in application frameworks. However, it must be possible for scripting in some cases to be very light, perhaps just specifying milestones tracked during the course of a method. But it also must be possible to add more detail on top of this, such as a practice that defines specific work products associated with method milestones or the inclusion of specific activities defined using more detailed practice scripts.

There is also an important distinction to be made between the rigorous scripting required for executable software and the more flexible scripting that must be allowed for methods. In the end, it is really the project team that enacts a method, and such teams are not "programmed" like computers. A method should be scripted in a way that is easy for practitioners to read and understand as guidelines for carrying out the method, while still giving them the freedom to use their judgment and do what makes sense within the context of their project.

Nevertheless, automated support for enactment can be very useful, especially for larger project teams, and the language must be defined precisely enough itself to allow for the automated support of methods formalized using the language. Such automation may involve the actual "execution" of method scripts, so that development team members can be given appropriate support for carrying out the process and their progress in doing so can be tracked.

However, the intent, in the end, is always to support the work of the development team, as the team sees most fit. It is the *method* that is being scripted, to whatever extent is appropriate, not the actual actions of the team. In fact, the very goal of the foundation being requested is to allow a development team to take control of its own development method, not to be controlled by it.

6.2.3.1 Users and usages of the Language

The Language has two major user types: practitioners and method engineers. This RFP primary target group is the practitioners, but method engineers also have an important role in defining methods.

Practitioners are primarily interested in the following usages of the language: read, evaluate, compare, compose, tailor, use and adapt. Method engineers are primarily involved in: define, compare and compose.

Practitioners want to use ready to use complete practices that each give a measurable value to the method, that is designed to be lean and that includes the verification of the work performed when using the practice. Practitioners must be able to reuse such complete practices, tailor them and adapt them based on retrospectives made by the team.

Method engineers create these complete practices by reusing practice fragments, which the method engineers glue together and fine tune to become a useful concrete practice. Method engineers also create method frameworks including packages of practices which are composed and from which a practitioner can select a subset to be used for a particular endeavor.

There will be practitioners at different competence levels. A very high percentage of the practitioners in the world are not interested in the definition of methods and practices. Their interest is to develop software only and thus to use the practices. Other more advanced practitioners are interested in having more detailed knowledge of the practices and they also want to have deeper understanding of the progress of their endeavor in producing the outcome of the project. Some, much fewer in today's situation, are interested in getting detailed support for the practices and its detailed activities.

6.3 Relationship to Existing OMG Specifications and activities

6.3.1 Software and System Process Engineering Metamodel (SPEM)

SPEM is an existing OMG specification that defines a framework, a metamodel and a UML profile to describe software and system engineering processes [SPEM2]. SPEM is specifically "targeted at process engineers, project leads, project and program managers who are responsible for maintaining and implementing processes for their development organizations or individual projects" [SPEM2, Subclause 6.2], and developers are primarily seen as end-users of the models defined by these. While SPEM specifically supports "deployment of just the method content and process needed by defining configurations of processes and method content" and "libraries of reusable process and method" [SPEM2, Subclause 6.2] through "clear separation of method content definitions from the development process application of method content" [SPEM2, Subclause 6.3.1], it does not provide a common kernel for that content. SPEM itself does not, therefore, provide the complete foundation requested by this RFP.

On the other hand, the SPEM 2.0 Profile [SPEM2, Clause 7] could provide a starting point for defining the language required by this RFP. Nevertheless, there are some fundamental issues that would need to be addressed in doing this.

For example, while the SPEM 2.0 Base Plug-in includes a concept of "Practice" [SPEM2, Subclause 18.3.8] that is similar to the way the term is used in this RFP, SPEM currently defines this concept simply as a kind of "guidance" in a method plug-in outside the core of the specification. In contrast, the language requested in this RFP is required to have "practice" as a foundational language construct used in the creation of methods.

Further, the work that led to SPEM started some time ago, before the rise of much of the recent agile and lean movement. Over the past ten years, we have come to better understand software practices and how to represent them in support of practitioners doing their tasks. Recent experience indicates that newer, innovative language

constructs will be necessary in order to meet the requirements in this RFP for the definition and use of the kernel in the creation of practices and methods (see, for example, the discussion of Semat in Section 6.4.2 below).

Given these considerations, this RFP does not require that the requested foundational kernel and language be based on or be a revision of SPEM. Nevertheless, the RFP allows for the possibility of a submission that does so build on SPEM. It would, of course, be incumbent on a submitter proposing using SPEM in this way to show how this would be effective in meeting the RFP requirements, just as it would be similarly incumbent on a submitter proposing a different approach.

6.3.2 Architecture Ecosystem SIG (AESIG)

The Architecture Ecosystem SIG [AESIG] is a special interest group organized under the OMG Architecture Board. The mission of the Architecture Ecosystem SIG is to work with OMG domain and platform task forces, other relevant OMG SIGs, external entities and related industry groups to facilitate the creation of a *common architectural ecosystem*. This ecosystem will support the creation, analysis, integration and exchange of *information* between modeling languages across different domains, viewpoints and from differing authorities.

The aim of the ecosystem is to allow languages to be more modular, and allow the domain architect to be able to flexibly tailor and integrate these modular languages to create viewpoints that are appropriate for their domain of interest. This ecosystem will be the natural successor to the current MOF and UML profiling standards. Depending on the RFPs that result from the AESIG effort, submissions to this RFP may wish to consider relevant results in their definition of the required language.

6.3.3 Case Management Process Modeling (CMPM) RFP

The Case Management Process Modeling RFP [CMPM] solicits proposals for a metamodel extension to BPMN 2.0 to support modeling of case management processes. Case Management focuses on actions to resolve a case – a situation to be managed toward objectives. Cases do not have predefined processes for achieving objectives. Humans make decisions based on observations, experience and the case file. Changes in the state of the case will result in new actions. A practice or discipline may adopt rules to guide decisions and make processes more repeatable. New modeling paradigms are required to facilitate all this.

Case management is typically suited to manage knowledge work, in particular work that is associated with innovation activities and initiatives. Integration with CMPM could be one possible approach considered by submitters to this RFP to support enactment of methods. There is an opportunity to seek alignment with the CMPM specification where possible. However, in order to allow for different approaches for a domain-specific language, this RFP is not mandating the foundation for this to be based on CMPM.

6.3.4 Software Metrics Metamodel (SMM)

The SMM specification defines a metamodel for representing measurement information related to software, its operation, and its design. The specification is an extensible metamodel for exchanging software-related measurement information concerning existing software assets (designs, implementations, or operations). A standard for the exchange of measures is important given the role that measures play in software engineering and design. The SMM is part of the Architecture Driven Modernization (ADM) roadmap and fulfills the metric needs of the ADM roadmap scenarios as well as other information technology scenarios.

6.4 Related non-OMG Activities, Documents and Standards

6.4.1 ISO/IEC 15288, ISO/IEC 24744 and ISO/IEC 12207

ISO/IEC 24744, Software Engineering Metamodel for Development Methodologies (SEMDM), establishes a formal framework for the definition and extension of development methodologies for information-based domains (IBD). This standard is intended to be used by method engineers while defining or extending development methodologies. ISO/IEC 12207 and ISO/IEC 15288 are related standards that only deal with process aspects of methodology and also focus on the method engineer. This RFP, on the other hand, is focused on the end-to-end practices, and the enactment stage conducted by software practitioners as described above in Section 6.2.

The ISO/IEC 24744 metamodel has been used by method engineers within the Situational Method Engineering (SME) community to define light-weight and flexible method approaches. The metamodel builds upon a small set of agreed core elements. The essential process components are stages, producers, work units and work products. The community has recently come to acknowledge two different characteristics of method enactment: (1) *tailored process*, essentially a static viewpoint, and (2) *performed process*, a more dynamic performance. This could provide a basis for a submission to this RFP.

6.4.2 Software Engineering Method and Theory (Semat)

Semat is an effort to “refound software engineering based on a solid theory, proven principles and best practices.” The Semat *Grand Vision* (the call for action statement) is supported by three dozen well-known individuals who have made significant contributions to the software community, by a dozen major corporations and by more than 1400 supporters from all over the world.

The work that has started within the Semat community has been a major influence on this RFP, and the fundamental organization of the requested foundation into a kernel and a language has been adopted from this work. However, the need for the kernel and language being requested by this RFP has been justified in Sections 6.1 and 6.2 above independently of the wider Semat vision. One reason for issuing this OMG RFP is

specifically to solicit participation beyond the current Semat community. Nevertheless, the ongoing experience of the Semat group provides some insight into ways in which a kernel and language such as requested in this RFP may need to go beyond what his been traditionally available for method and process definition.

For example, it was realized early in the Semat effort that, rather than just providing the means to define work products to be produced, it would be necessary to capture in the kernel a conceptualization of tracking project state at a higher level than the specific work products produced by any specific practice or method. This led to the introduction of *Abstract-Level Project Health Attributes,* or "Alphas", that "subsume and encapsulate work products at a higher level of abstraction." An Alpha is defined as "a property of the current state of a software project, satisfying the following criteria:

- It is relevant to an assessment of the project's health, that is to say, of the degree to which the project satisfies its stated objectives (such as deadlines, costs, quality).
- It can be determined, directly or indirectly, in terms of the current state of the project's work products. The indirect case means that the definition of an alpha may involve work products as well as previously defined alphas."

Of course, it is not required that a submission to this RFP adopt this concept of an Alpha in order to structure the proposed kernel. However, the concept is an example of how, at least in the experience of the Semat community, some innovation will be necessary in order to create a proper kernel that goes beyond previous approaches to method content definition.

The Semat effort will continue outside the scope of this RFP, in its current capacity as a self-organizing community. The usefulness of the result of the RFP to the general software development community, though, is not dependent on the future work of Semat, or even its continuation at all. Nevertheless, it is expected that an adopted OMG specification for the requested kernel and language will be highly synergistic, by intent, with the continued realization of the Semat vision.

6.4.3 Eclipse Process Framework (EPF) and Unified Method Framework (UMF)

The open-source Eclipse Project Framework (EPF) project (http://www.eclipse.org/epf) under the Eclipse Foundation represents an important community in existence today that has implemented tool support and practices according to the existing SPEM specification [SPEM]. EPF aims at producing a customizable software process engineering framework, with exemplary process content and tools, supporting a broad variety of project types and development styles.

EPF implements the Unified Method Framework (UMF)[1]. The UMF framework defines a method plugin architecture and provides a set of core infrastructure method elements, i.e. the core method plugins. The framework and its core method elements

[1] http://epf.eclipse.org/wikis/epfpractices/practice.bus.mdev.base/guidances/concepts/umf_62CAA5FA.html

can be perceived as a practice-agnostic "EPF kernel" that can be used to define a wide variety of practices.

EPF comes with a set of defined practices[2] that is intended to be used by process engineers to learn about the practices in order to make decisions about which practices to include in a process configuration. EPF is an implementation of the SPEM 2.0 standard that has tried in this way to address some of the goals of this RFP. It may be possible to use this experience to develop a submission for this RFP along similar lines.

6.5 Mandatory Requirements

6.5.1 The Kernel

6.5.1.1 Domain model

The Kernel shall be represented as a domain model of a small number (expected to be closer to 10 than a 100) of essential concepts of software engineering and their relationships. The Kernel shall be expressed in the Language. (In the following when referring to the Language, we mean the language requested in this RFP.)

6.5.1.2 Key conceptual elements

The Kernel shall define the key conceptual elements that all software engineering endeavors have to monitor, sustain and progress, covering at least the following kinds of concepts (the specific grouping used here is not required):

a. *System:* Concepts related to the system being produced, for example: software, platform, etc.

b. *Functionality:* Concepts related to the required function of the system being produced, for example: requirements, needs, opportunities, stakeholders, etc.

c. *People:* Concepts related to the people required to create a system with the required functionality, for example: project, team, role, etc.

d. *Way of Working:* Concepts related to the way an organized team carries out its work to create a system with the required functionality, for example: method, practice, goal, etc.

6.5.1.3 Generic activities

The Kernel shall define the generic activities that a team will need to undertake to successfully engineer and produce a software system, covering at least the following kinds of activities (the specific grouping used here is not required):

[2] http://epf.eclipse.org/wikis/epfpractices/index.htm

a. *Interacting with stakeholders:* Activities related to necessary interactions with stakeholders, for example: exploring possibilities, understanding needs, ensuring satisfaction, handling change, etc.

b. *Developing the system:* Activities related to actually constructing a system, for example: specifying, shaping, implementing, testing, deploying and operating the system.

c. *Managing the project:* Activities related to managing a project, for example: steering the project, supporting the project team, assessing progress and concluding the project.

6.5.1.4 Kernel elements

The definition of each element of the Kernel shall include the following:

a. A concise description of the meaning of the element and its use in software engineering, intuitively understandable to a practitioner.

b. The relationships of the element to other elements in the Kernel.

c. The various different states the element may take over time, including initial/entry and final/exit criteria as appropriate for the element.

d. How the element is applied in practice, including how it may be instantiated, tailored, extended, etc., to support the work of a specific project team using specific practices.

e. How different ways of applying the element may be compared to each other and guidance on deciding among the alternatives.

f. Appropriate metrics that can be used to assess progress, quality, etc.

6.5.1.5 Scope and coverage

The Kernel shall be sufficient to allow for the definition of practices and methods supporting projects of all sizes and a broad range of lifecycle models and technologies used by significant segments of the software industry.

6.5.1.6 Extension

The Kernel shall also allow for extension, both in terms of addition of new elements and providing additional detail on existing elements that provide for practice-specific work products.

a. The Kernel shall allow for project and organization specific extensions.

b. The Kernel shall be tailorable to specific domains of application and to projects involving more than software, e.g., to serve as a basis for future extensions for systems engineering.

6.5.2 The Language

6.5.2.1 Language Definition

6.5.2.1.1 MOF metamodel

The Language shall have an abstract syntax model defined in a formal modeling language. The submission is expected to reflect this requirement in a description or mapping to the OMG architectural framework based on MOF.

6.5.2.1.2 Static and operational semantics

The Language shall have formal static and operational semantics defined in terms of the abstract syntax.

6.5.2.1.3 Graphical syntax

The Language shall have a graphical concrete syntax that formally maps to the abstract syntax. The submission is expected to reflect this requirement in a description following the Diagram Definition specification [DD] unless arguments are given for choosing something else.

6.5.2.1.4 Textual syntax

The Language shall also have a textual concrete syntax that formally maps to the abstract syntax.

6.5.2.1.5 SPEM 2.0 metamodel reuse

Proposals shall reuse elements of the SPEM 2.0 metamodel where appropriate. Where an apparently appropriate concept is not reused, proposals shall document the reason for creating substitute model elements.

6.5.2.2 Language Features

6.5.2.2.1 Ease of use

The Language shall be designed to be easy to use for practitioners at different competency levels:

a. Those that have very little modeling experience and quickly and intuitively need to understand and learn how to use the Language.

b. Intermediate users who are more advanced and willing to describe what kind of outcome they expect of their work.

c. Advanced users that can work with all aspects of the Language to model their complete software endeavor.

6.5.2.2.2 Separation of views for practitioners and method engineers

The Language shall provide features to express two different views of a method: the method engineer's view and the practitioner's view. The primary users of methods and practices are practitioners (developers, testers, project leads, etc.).

The proposal shall be accessible to both practitioners and method engineers, but should target the practitioners first and foremost. Extensions should support method engineers to effectively define, compose and extend practices, *without complicating* its usage by the practitioners.

6.5.2.2.3 Specification of kernel elements

The Language shall have features for specifying Kernel elements, including:

a. Formal and informal descriptions of the content and meaning of an element.

b. The relationship of the element of other elements.

c. States the element may take over time and the events that cause transitions among those states.

d. How the element is instantiated, including provisions for practice-specific tailoring of the element, and the basis for comparing different instantiations.

e. Metrics defined to assess various attributes of the use of the element.

6.5.2.2.4 Specification of practices

The Language shall features for specifying practices in terms of Kernel elements, including:

a. Description of the particular cross-cutting concern addressed by the practice and the goal of the application of the practice.

b. The Kernel elements relevant to the practice and how they are instantiated for use in the practice, including any practice-specific tailoring of the elements.

c. Any work products required by and produced by the practice.

d. The expected progress of work under the practice, including progress states, the rules for transition between them and their relation to the states of relevant Kernel

elements used in the practice. (For example, describing a practice that involves iterative development requires describing the starting and ending states of every iteration.)

e. Verification that the goal of the practice has been achieved in it application, particularly in terms of measurements of metrics defined for its elements.

6.5.2.2.5 Composition of practices

The Language shall have features for the composition of practices, to describe existing and new methods, including:

a. Identifying the overall set of concerns addressed by composing the practices.

b. Merging two elements from different practices that should be the same in the resulting practice, even if they have different contents defined in the practices being composed. (For example, a use case practice may have a work product called Use Case, with a name, a basic flow etc. A testing practice may have work product called Testable Requirement with an identifier and a description. In the method resulting from composing these two practices, these two work products should be merged into one, where the name of the Use Case is the identifier of the Testable Requirement and the basic flow of the Use Case is the description of the Testable Requirement).

c. Separating two elements from different practices that should be different in the resulting practice, even though they may superficially seem to be the same. (For example, in a testing practice there may be a work product called Plan and in an iterative development practice there may also be a work product called Plan. In the method resulting from composing these two practices these two work products must be different – e.g., the Testing Plan vs. the Development Plan.)

d. Modifying an existing method by replacing a practice within that method by another practice addressing a similar cross-cutting concern.

6.5.2.2.6 Enactment of methods

The semantic definition of the Language shall support the enactment by practitioners of methods defined in the Language, for the purposes of

a. Tailoring the methods to be used on a project

b. Communicating and discussing practices and methods among the project team

c. Managing and coordinating work during a project, including modifications to the methods over the course of the project by further tailoring the use of the practices in the method

d. Monitoring the progress of the project

e. Providing input for tool support for practitioners on the project.

6.5.3 Practices

The focus of this RFP is on the Kernel and the Language that will be used to describe software engineering practices. Submissions are expected to demonstrate non-normative examples using the Kernel and the Language as defined in the submission.

6.5.3.1 Examples of Practices

f. Submissions shall provide working examples to demonstrate the use of the Kernel and Language to describe practices. Preferably these examples should be drawn from existing and well-known practices.

g. Submissions shall provide working examples to demonstrate the composing of practices into a method.

h. Submissions shall provide working examples to demonstrate how a method can be enacted.

i. Submission shall include a capability to demonstrate the operational execution of methods as a proof of concept.

It is expected that the example practices are well-structured and suited to demonstrate how well the proposed Kernel and Language can be used to define good-quality practices. Each example of practice shall:

a. be described on its own, independent from any other practice

b. be either explicitly defined as a continuous activity or have a clear beginning and end states

c. bring defined value to its stakeholders

d. be assessable; in other words, its description must include criteria for its assessment when used

e. include, whenever applicable, quantitative elements in its assessment criteria; correspondingly, the description must include suitable assessing metrics.

6.5.3.2 Existing Practices and Methods

Respondents shall provide a guideline for how existing SPEM-based practices and methods, and possibly other representations, can be migrated to the new proposed specification.

6.6 Optional Requirements

None

6.7 Issues to be discussed

a. Submissions shall include the definition of the Kernel along with any alternative options considered for the names and definitions of Kernel elements, and the reasons for not using such alternatives.

b. Submissions not based on SPEM 2.0 should discuss why they did not use SPEM and clearly describe and demonstrate the main differentiators.

6.8 Evaluation Criteria

6.8.1 Objective

Submissions will be evaluated based on the following general criteria.

a. Support of the principal goal for the improvement of software and system products and methods.

b. Inclusion of only the essentials of software engineering.

c. Grounding of Kernel and Language on a solid theoretical basis

d. Ability to be applied to different scales of projects, from small to very large projects.

e. Justification by a clear rationale.

f. Ability to support agile/lean software development methods as well as traditional prescriptive methods. (It is imperative that this flexibility should not lead to increasing the complexity in the Language or the Kernel, rather, it is an integral part of them.)

6.8.2 Kernel and Language

The proposed Kernel and Language will be evaluated based on the following criteria.

a. Applicability to all software engineering efforts.

b. Ability to contribute in a positive and perceptible way to the quality of software products. Although the evaluation may take multiple years, it should be evaluated by comparing different approaches and making an objective assessment.

c. Applicability to all software engineers, regardless of their backgrounds, and their methodological camps (if any).

d. Precision of the definition of Kernel elements.

e. Applicability of Kernel elements, through precise guidelines that projects can apply.

f. Suitability of Kernel elements for quantitative evaluation of their application.

g. Comprehensiveness in capturing the Essence of Software Engineering, providing a Kernel with its essential elements that supports the crucial ways of working of software engineering teams.

h. Ability of experienced practitioners to use the ideas presented in the Kernel to execute a software endeavor without the need for further explicit guidance. (That is the Kernel with its essential elements should be useful in itself without the composition of practices.)

i. Ease of use by software practitioners. Descriptions written in the Language should be easy to understand by all its users. The Language should be designed for the developer community, not just process engineers and academics.

j. Coverage of relevant practices and their composition in today's methods.

k. The ability of guidance on practices and methods, as defined in the Language, to be queried, such that a practitioner can easily discover relevant guidance and have it presented in new and informative ways.

l. The descriptions are built in terms of the essential elements of the Kernel helping one of the principal goals: to avoid reinventing practices and methods.

m. Support for simulating the application of methods and practices, thus providing insight into the dynamics of the method.

n. Support for applying methods and practices (as described through the Language) in real projects.

o. Support for the comparison of methods and practices to see which are suitable for a given situation.

p. Support for assessing whether a project claims to apply a given method or practice (as described through the Language) actually does. (This problem is sometimes

referred to as "closing the gap" between what project teams say they do and what they actually do.)

q. Support for the measurement of practices and methods, both to enable performance evaluation and to guide evaluation and validation in research.

r. Ease with the way in which practices defined through the Language can be taught.

s. Support for quantitative assessment of all the relevant artifacts.

t. Support for mechanisms enabling the evolution of the Kernel.

u. The ability to add practices, levels of detail and lifecycle models.

6.8.3 Practices

Examples of practices will be used to evaluate the proposal. Specifically, to evaluate how well they demonstrate positive qualities of the proposed solution in terms of the above Kernel and Language evaluation criteria.

6.9 Other information unique to this RFP

NONE

6.10 RFP Timetable

The timetable for this RFP is given below. Note that the TF or its parent TC may, in certain circumstances, extend deadlines while the RFP is running, or may elect to have more than one Revised Submission step. The latest timetable can always be found at the OMG Work In Progress page at http://www.omg.org/schedules under the item identified by the name of this RFP. Note that "<month>" and "<approximate month>" is the name of the month spelled out; e.g., January.

Event or Activity	Actual Date
Preparation of RFP by TF	*ADTF – December, 2010 – June, 2011*
RFP placed on OMG document server	*"Four week rule" – May 23, 2011*
Approval of RFP by Architecture Board *Review by TC*	*June 23, 2011*
TC votes to issue RFP	*June 24, 2011*
LOI to submit to RFP due	*November 22, 2011*
Initial Submissions due and placed on	*February 22, 2012*

OMG document server ("Four week rule")	
Voter registration closes	*February 22, 2012*
Initial Submission presentations	*March 21, 2012*
Preliminary evaluation by TF	*March 21, 2012*
Revised Submissions due and placed on OMG document server ("Four week rule")	*August 15 , 2012*
Revised Submission presentations	*September 12, 2012*
Final evaluation and selection by TF *Recommendation to AB and TC*	*December 5, 2012*
Approval by Architecture Board *Review by TC*	*December 6, 2012*
TC votes to recommend specification	*December 7, 2012*
BoD votes to adopt specification	*December 7, 2012*

Appendix A References and Glossary Specific to this RFP

A.1 References Specific to this RFP

The following documents are referenced in this document:

[AESIG] "Architecture Ecosystem SIG", http://www.omgwiki.org/architecture-ecosystem/doku.php

[DD] "Diagram Definition, Version 1.0 – FTF Beta 1", OMG Document ptc/2010-12-18, http://www.omg.org/spec/DD/1.0/Beta1/

[BPMN2] "Business Process Model and Notation, Version 2.0", OMG Document formal/2011-01-03, http://www.omg.org/spec/BPMN/2.0/

[CMPM] "Case Management Process Modeling (CMPM) RFP", OMG Document bmi/09-09-23, http://www.omg.org/cgi-bin/doc?bmi/09-09-23

[Semat] "Software Engineering Method and Theory – A Vision Statement", http://www.semat.org/pub/Main/WebHome/SEMAT-vision.pdf

[SMM] "Architecture-Driven Modernization (ADM): Software Metrics Meta-Model (SMM) – FTF Beta 1", OMG Document ptc/2009-03-03, http://www.omg.org/spec/SMM/1.0/Beta1

[SPEM1] "Software Process Engineering Metamodel (SPEM) Specification, Version 1.0", OMG Document formal/02-11-14, http://www.omg.org/cgi-bin/doc?formal/02-11-14

[SPEM2] "Software & Systems Process Engineering Metamodel (SPEM) Specification, Version 2.0", OMG Document formal/2008-04-01, http://www.omg.org/spec/SPEM/2.0/

A.2 Glossary Specific to this RFP

Here is a list of some key terms specific to this RFP as they have been used in the RFP text. It is expected that a submission will contain a more comprehensive glossary of terms, and also might update these definitions if deemed necessary.

Method - A *method* is a systematic way of doing things in a particular discipline. For the purpose of this RFP, the relevant discipline is software engineering.

Practice - A *practice* is a general, repeatable approach to doing something with a specific purpose in mind, providing a systematic and verifiable way of addressing a particular aspect of the work at hand. It should have a clear goal expressed in terms of the results its application will achieve and provide guidance on what is to be done to achieve the goal and to verify that it has been achieved. Such practices may include specific approaches for software design, coding, testing at various levels, integration, organizing and managing the development team, etc. Practices are being defined using elements from the Kernel (see below).

Kernel - The Kernel includes essential elements of software engineering. The Kernel represents a *domain model* for software engineering that provides a common terminology of concepts and their relationships that may be used in the definition of software engineering practices. The Kernel is defined in terms of the Language (see below).

Language - In this document the Language is the standard modeling language requested by this RFP for specifying practices based on the Kernel and for composing methods from the practices. It can be used by a development team to both informally discuss and sketch their methods and then formalize those methods as they find appropriate. The Language has an abstract syntax, static and operational semantics. It also has a concrete lexical syntax and a concrete graphical syntax.