PIT-ProcessM: A Software Process Improvement Meta-model

Paula Ventura Martins  
INESC-ID, Universidade do Algarve  
Campus de Gambelas,  
8005-139 Faro, Portugal  
pventura@ualg.pt

Alberto Rodrigues da Silva  
INESC-ID /Instituto Superior Técnico  
Rua Alves Redol, nº 9  
1000-029 Lisboa, Portugal  
alberto.silva@acm.org

Abstract

Software Process Improvement (SPI) is one of the main actual software development challenges. Process meta-models allow capturing informational and behavioural aspects of software development processes. Unfortunately, standard process meta-modelling approaches, such as the Software Process Engineering Meta-model (SPEM), OPEN Process Framework (OPF) and Standard Meta-model for Software Development Methodologies (SMSDM), focus just on process description, providing different models for several versions of the same process. According to these meta-modelling approaches, it is not possible to compare and identify improvements in an improved process. This lack of information recognizes that further research in SPI meta-model is needed to reflect the evolution/change on software processes. Considering this limitation in SPI meta-modelling, this paper presents a comparative study of the most recognized process meta-models approaches and introduces a new SPI based meta-model designed by ProjectIT-Process Meta-model (PIT-ProcessM). Our intention is to present observed problems in existing approaches and propose a process meta-model that addresses features related to process changes and evolution.

1. Introduction

Software process improvement (SPI) is a challenge to organizations trying to continually improve software quality and productivity and to keep up their competitiveness [1]. Organizations tend to react to: (1) changes in the environment that they operate, (2) changes at a corporate level, (3) unplanned situations not considered in the model, or (4) improve the quality of their final products. Such changes may be caused, for example, by poor performance, by new tools acquired by the company to support its software development teams, changes in the marketing strategy or in clients’ expectations and requirements. Thus, an existing process model must be modified or extended to reflect the evolution of the environment and/or internal changes. However, existing process models which only take into account descriptive aspects, such as work related activities and technical work products, can’t address such features.

Before thinking about SPI, previously we provide some support to define and describe processes themselves. Descriptive process modelling aims to describe the actually performed development process, especially because of the wide range of existing processes. The models are used as the basis for understanding and analyzing processes, improving existing processes, as a baseline for process changes or for disseminating process knowledge [2]. While process modelling has a long history, many authors [3] express concern about the inadequacies of them which, among other things, do not offer any support to represent and identify changes in a process in order to know whether or not a changed process is improved. In the last decade, meta-models have been proposed as a means of creating additional rigour for process modelling [4, 5]. These meta-models are useful to specify concepts, rules and in general relationships used in the process definition. So, processes defined by meta-models offer a higher degree of formalization and better support for consistent extensions or modifications; the main goal of SPI.

This paper contribution is to present not only a new SPI meta-model, but also to discuss the advantages and disadvantages of existing process meta-models comparatively to ProjectIT-Process Meta-model (PIT-ProcessM). PIT-ProcessM is a meta-model approach based on the OMG four-layer architecture that addresses process models version comparison to validate improvements in process models.

An outline of three well known meta-models intends to depict only the strengths and weakness of each approach to justify the creation of our meta-model proposal. One such meta-model is the Software Process Engineering Meta-model (SPEM) from the Object Management Group (OMG) [6]. The SPEM is based on the OMG Meta Object Facility (MOF) [7] and the current version is SPEM v2.0. OPEN Process Framework (OPF) [8] was originally created in the mid-1990s based on several earlier object-oriented software development processes. Standard Meta-model for Software Development Methodologies (SMSDM) [9] supports not only process but also capability assessment.
This paper is organized in the following sections. Section 2 describes briefly key concepts from the proposed PIT-ProcessM. Section 3 outlines version management in PIT-ProcessM. Section 4 presents a critical analysis of other software process meta-models and also includes a comparison with PIT-ProcessM. Finally, Section 5 concludes and discusses future trends.

2. PIT-ProcessM – Key Concepts

The PIT-ProcessM meta-model describes formal process elements that can be applied to construct software development processes. Figure 1 intends to give an overview of the proposed meta-model and its views. PIT-ProcessM includes two complementary views: the static view and the dynamic view. The static view shows the structure of process concepts, their characteristics and the relations between them. The dynamic view identifies a structure that is specific to temporal development circumstances such as how work is to be organized over time.

Figure 1 illustrates the elements (meta-classes) of the PIT-ProcessM meta-model that are relevant for modelling functional aspects (activity, discipline, process, iteration and phase), informational aspects (work product and work product kind) and organizational aspects (role). These basic elements are:

- **Process** is made up of a number of process elements (several disciplines or other existing processes).

- **Discipline** is a particular organization of process activities according to a common “theme”. Each discipline includes several practices which are described through its Activities, WorkProducts and Roles.

- **Activity** is performed to produce (e.g., create, evaluate, iterate, and maintain) WorkProducts. An Activity is owned by a Role. Beside the owner, there are additional Roles that also participate in the activity.

- **Phase** is the time interval that provides a macro organization of Activities to be performed and it has the main goal, to produce a final set of work products.

- **Iteration** is a scheduled part of a Phase leading to an increment towards the final phase goal. A Phase can have one or more Iterations.

- **WorkProduct** is a significant element (e.g., document, diagram, model and application) that is produced or consumed by process Activities.

- **WorkProductKind** represents several types of work products. WorkProducts can be classified in several types which identify the kind of input or/and output expected in an activity (such as text documents, UML models, SQL tables, executables, code libraries, and so on).

- **Role** performs one or more Activities in order to produce, either directly or indirectly, versions (or new) of one or more WorkProducts.

3. PIT-ProcessM – Version Support

Research on software process meta-models intends to represent process elements that constitute good practices. Nevertheless, also important is the way processes evolve within the changing needs of the development organizations. Existing process meta-models lack of support on process evolution in order to
effectively improve and assess software process. A process model has to be modified or extended to reflect the evolution of the environment and/or organizational changes. However, existing process stay independent from it once created or, worst, older version disappears. One of the most challenging issues in SPI is related to the fact that existing meta-models do not allow representing changes in process models and, at the same time, keep reference to previous versions in order to evaluate the success of the new changes introduced in the process.

Typically, processes are constantly changing, when a process is modified, effects on the process model as well as on existing derived projects have to be considered. In particular, it has to be ensured that process model modifications do not affect the correctness of the process model and existing projects. One solution to prevent invalid projects due to process model modifications is to disallow entirely model updates when a project is active. However, this is not a viable solution, because we propose an iterative and dynamic approach, i.e., the modification of process models while one or more projects are still active. So, based on these assumptions, some conditions have to follow: (1) the process model is modifiable and (2) both, projects and process models must remain consistent with each other.

PIT-ProcessM includes concepts for version management (Figure 1b). The basic idea proposed in the new approach to versioning process models is not to update the process model in place, but to version them. Consequently, process elements to be put under version control are activities, disciplines, work products, etc. (Figure 1b). When a process model is created, this includes the creation of a first version, called root version. New versions derived from existing ones through several operations (Operation meta-class) performed, such as, create, update and delete of elements. In version management models, when a new version of an element is created, it is related with related only with its predecessor by the version_of relationship and the element version meta-class that will be associated with an Operation.

However, a Process is a complex structure composed of other objects, such as, other Processes or a group of Disciplines. On the other hand, this composite structure constitutes a hierarchy. When the creation of a version of an element is requested a bottom-up method will take place to keep consistency between versions of different elements (process, disciplines, etc.). After the creation of an element version, the bottom-up method climbs the composition structure to the version of the composite element at the highest level (Process). Version propagation is an automatic approach that incorporates new versions of the composite elements each time a version of one of its elements is created. Therefore, each time the element is updated, deleted or created, a version of highest level elements is updated, deleted or created. This approach has an advantage that no additional effort is needed in identifying differences between a version and its ancestors (no comparison needed).

A Process meta-class includes an unique identifier (process name) and a process version tree. A process version defines a version number and it is either in state transient, released or obsolete. Versions associated to a Process form a version tree, where parent-child relationships in the tree represent ancestor-descendant (old version - new version) relationships.

During a process version lifetime, a version is in one out of a set of versions states, namely: transient, released and obsolete (Figure 2). Initially, a root version is in transient state. In the case of a version derived from an existing one, an inheritance of the existing version is taken and the new version is put into transient state. In the transient state, a version may be updated or deleted by applying modification operations to it. In order to prevent invalid processes, when a version is in state transient is not allowed: (1) to create descendants versions; (2) to create projects based on that version; and (3) to be referenced by a composite process. When the update actions of a transient version are finished, the state of the version is changed to released.

![Figure 2. Version management](image-url)

In the released state, a process version may not be deleted or updated, but projects based on it can be specified. It is also possible to be referenced in a composite process version and descendants of it can be created. When a released version has to be modified, its state is changed to transient, but only in special conditions: (1) no process versions are derived from it; (2) no projects are defined based on it; and (3) it is not referenced in a composite process version. If a released
version becomes obsolete and no project is specified
based on it, its state is changed to obsolete. In case a
version is in obsolete state and has no descendants, no
active projects and it is not referenced by a composite
process, the state of the process version may be changed
to the final state in which the version will be deleted.

4. Related Work

The purpose of this section is to give a short
overview of some of the most widely recognized process
meta-modelling approaches to illustrate problems related
to SPI. Such approaches are: Software Process
Engineering Meta-model (SPEM) [6], OPEN Process
Framework (OPF) [10] and Standard Meta-model for
Software Development Methodologies (SMSDM) [9].

SPEM 2 had been defined by the Object Management
Group as a high-level standard for representing
organization’s processes and expertise [6]. This meta-
model has been specified as a MOF 2 model [7] and
reuses some key classes from the UML 2 Infrastructure
[11]. SPEM 2 also defines a UML 2 Superstructure-
based Profile (referred to as the ‘SPEM 2.0 Profile’).

This meta-model is structured into seven packages.
Nevertheless, a pure structural analysis of SPEM 2
comprises only two meta-model packages: Core and
ProcessStructure. The Core package contains classes
and abstractions that build the foundation for all other
meta-model packages. The ProcessStructure package
defines the base to represent basic process models in
terms of activities, work products and roles (RoleUse).

Figure 3.SPEM 2 meta-model overview [6]

One of the characteristics of SPEM 2 is the inclusion
and confusion between the notion of time (Phase and
Iteration) and that of work activities (Work Definition/
Activity). From SPEM 2 definitions and notation, it is
clear that the general notion of work activities is of a
“job to be done”. Nevertheless, according to the
Software Engineering Body of Knowledge (SEWBOK)
[12], the temporal structure is to be kept separate from
work-related notions. In SPEM 2, an ActivityKind does
not specify the time needed to “do a job” or when a
WorkProduct should be delivered. The ActivityKind
meta-class of the SPEM 2.0 Base Plug-in was specified
to provide the capability of defining lifecycle models.
However, it is included in the model as a predefined
special Activity. The notion of Phase and Iteration set
time limits to process elements. So, in SPEM 2, Phase
and Iteration (temporal concepts) are defined as a Kind
of Activity (ActivityKind), a non-temporal concept
(Figure 3).

Other technical aspects specific from SPEM 2 are
related with the definition of Process. Typically, the
word Process has been used as an all-inclusive term,
constituted by a number of process components. SPEM
2 models a Process as a sub-metaclass of
ProcessComponent (Figure 3). This concept goes
against accepted whole-part meta-modelling approaches.
In SPEM 2, there is no meta-class to model the
component parts of a process. Probably, the meta-class
ProcessComponent should be called ProcessElement
(similar to the UML 2 Profile).

Figure 4 depicts an excerpt of actual meta-classes in
the OPF [10] meta-model. OPF intends to give a more
full meta-level description of process-focused meta-
classes than SPEM 2. The meta-model is so complex
that is impossible to show in one diagram all the meta-
classes that are part of the meta-model (further details
can be found in [8]).

In OPF, Work_Product has several sub-metaclasses,
such as Applications and Business Model (showed in
Figure 4). Other sub-metaclasses not present in the
diagram are: Architectures, Requirements, Models,
Metrics, Diagrams, Documents, Data Component,
Software Components, and Hardware Components. OPF
temporal sequencing concept bears similarities to SPEM 2, however the connection to WorkUnit is not a generalization (as in SPEM 2) but an association via an abstract super-class called Stage. A “missing” meta-class (compared to SPEM 2) is the overall relationship between Process and Process_Component (as discussed previously).

To conclude, we do not argue with this kind of meta-model that integrates at same time process and project concepts. The meta-model complexity results from including, at process level, project concepts like Project and Programme. Other data such as Language and its sub-metaclasses, WorkProduct sub-metaclasses should be discarded from meta-model, leaving these details to be added at the process level. WorkProduct sub-metaclasses should be replaced by the metaclass WorkProductKind.

SMSDM [9] is a generic meta-model that encompasses aspects from SPEM 2 and OPF. Also important is the existence of a reference model to support the notion of capability levels for software process improvement. With this main goal, the proposal captures well-known best practices, but we do not agree with the fundamentals of this proposal. The main reason is the incorrect interpretation of OMG four layers architecture within the new context of model engineering. Meta-classes in the meta-model are used by process engineer to create instances (i.e. objects) in the process layer and thus generate a process. As they argue, the objects in the process layer are often used as classes by the development team to create objects in the project layer.

Figure 5. SMSDM excerpt overview [9]

Figure 5 overviews an excerpt of SMSDM meta-model, where the upper meta-classes represent elements in the process layer and lower meta-classes represent elements in the project layer. TemplateElement is the abstract super-metaclass of the same elements at project level. This pattern is refined into more specialized patterns formed by sub-metaclasses of TemplateElement and ProjectElement (StageKind/Stage, WorkUnitKind/WorkUnit, etc.).

This meta-model uses meta-class pairs to represent process and project level concepts, so it is not surprising that SMSDM includes more elements than any other meta-model. SMSDM meta-model describe only the most important aspects concerning process and project layers. Details of each of its aspects are omitted.

Table 1 summarizes the core concepts used to compare the four meta-models. In this comparison, we just focused across different concepts and mandatory issues related to SPI. SMSDM and PIT-ProcessM, contrary to SPEM 2 and OPF, include a minimal set of process modelling elements to describe the process, avoiding, at the same time, unnecessary structural constraints on the resultant processes.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEM 2</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>OPF</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>SMSDM</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>PIT-ProcessM</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

OPF and SMSDM provide a direct context to support project modelling. However, the fundamentals of these proposals do not agree with OMG four layers architecture within the new context of model engineering, stating that processes elements defined as objects in the model level are often used as classes to create objects at project level. SPEM 2 and PIT-
ProcessM provide greater flexibility and support than OPF and SMSDM, even though OPF and SMSDM provide more guidance and structure. SPERM 2 only models the process layer, most of its concepts map to ‘kind’ meta-classes in SMSDM and it doesn’t have facilities to model the project layer. Also visible is the fact that no suitable product concepts are found in it. The OPF meta-model integrates at the same level process and project concepts. However, there is no clear boundary between process entities and project entities. With SMSDM, both process components and project entities can be described directly by the same meta-model. However, SMSDM meta-model defines and separates clearly classes that represent elements in the process and the project layers.

The activity of modelling and specifying processes to develop a process model also requires a focus on the integration between processes (process composition). This topic had been addressed only by our proposal (PIT-ProcessM).

PIT-ProcessM is the only that ensure that a complete picture of process version management. Subsequently changes made to one process version will allow a comparative assessment with older versions. Nevertheless, measurement was not explicitly included in PIT-ProcessM. In future, PIT-ProcessM will include measurement meta-classes and respective relationships to measuring and managing processes. Support for CMMI goals is no yet supported in PIT-ProcessM. In a near future, CMMI integration with supporting processes and measurements will be part of our proposal.

5. Conclusion

We discuss existing process meta-models and present PIT-ProcessM as a new meta-model that includes process version management. We demonstrate that approaches like SPERM 2, OPF and SMSDM are not specific enough to catch the needs for SPI. Process meta-modelling techniques must identify the most appropriate concepts not only to represent process models but also to support process evolution and change.

The capability and maturity of an organization affects the dimension of their software development processes. Therefore, SPI reference models must be taken into account in the meta-model layer. Presented meta-models do not support the notion of capability levels. SMSDM describes capability levels in terms of purpose and outcomes. A mapping between the process of a target organization and a SPI reference model is essential not only to identify achieved goals but also to evidence future improvement areas.

Several and distinguish meta-model approaches had been developed to support software development processes and capability/maturity models. However, an integrated solution that supports needs and assessment of software processes related to capability/maturity levels was not yet achieved. However, this wasn’t the goal of this paper.

References


