An Analytical Review of Process-Centered Software Engineering Environments

Reza Matinnejad, Raman Ramsin
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
r_matinnejad@alum.sharif.edu, ramsin@sharif.edu

***Abstract*—Process-centered Software Engineering Environments, or PSEEs, are intended for the definition, modification, and enactment of software process models; they thus bring software development processes into effect. Even though research efforts in process-centered software engineering abound, PSEE technology has not received the attention that it deserves. In order to create a concise but effective and practically applicable evaluation framework for PSEEs, this paper first presents a survey of PSEEs and highlights the current state of the art of the technology. The PSEEs which have been reviewed herein have been regarded as software systems, and as such, have been characterized in terms of their requirements. After providing a conceptual critique of the scope and nature of conventional PSEEs, a detailed criteria-based evaluation of a select set of several recent PSEEs has been conducted. The evaluation criteria have been derived from PSEE requirements and the results of the critique, and have then been refined and evolved into the final criterion set.**

Keywords-software process; process modeling; process enactment; Process-centered Software Engineering Environment (PSEE);

I. INTRODUCTION

Responding to the urgent need for high-quality software is one main goal of software engineering. While there are several quality factors for software in the literature, evaluation of software product quality is not as straightforward as it may seem. The difficulty of software product quality assurance has motivated the indirect approach of injecting quality into software *products* by concentrating on the quality of the software development *process*. Experience has proved that there exists a direct correlation between the quality of the development process and the quality of the developed software. Consequently, we can gain more control over the required quality of software products by controlling software processes.

It has long been observed that "software processes are software too" [38]. Consequently, the notion of *process programming* has been introduced, dealing with software processes as pieces of software. In other words, we can describe, examine, design and even test software processes, just as we do for software itself.

A software process is typically defined as a set of step-by-step activities that must be carried out to pursue the aim of developing a software product. A precise specification of these steps can be expressed using a *Process Modeling Language (PML)*. Since the advent of process programming, several PMLs have been introduced in the literature [22]. However, the OMG's SPEM, which is a standard meta-model used to specify models for processes, seems to have put an end to all the debates [8].

A *process model* is an explicit representation of a software process in a PML. A process model can be thought of as a vehicle for indicating how to carry out software development activities, how to specify the roles and tasks of software developers, and how the actions of developers might be supported by software tools and vice versa. It should tell developers how and when to activate automated tools, and how and when to give them feedback from process execution. Putting a process model into effect is called *process enactment*. Software engineering practices can be applied, guided and automated through process enactment.

A *Process*-Centered *Software Engineering Environment (PSEE)* is an environment that provides various services for software developers by enacting process models. For example, interactive assistance throughout software development, automation of routine and labor-intensive tasks, invocation and control of software development tools, and enforcement of mandatory rules and practices are all among the typical features of PSEEs. PSEEs go by many names: Process Sensitive Environments (PSE), Process Support Systems (PSS) and Process Centered Environments (PCE) are some of the alternative terms used in the literature for referring to PSEEs. PSEE Technology, however, should be distinguished from related technologies such as Workflow Management (WFM) and Computer-Supported Cooperative Work (CSCW). While PSEEs target the management of software development processes, WFM systems are mainly concerned with modeling and automation of business workflows and industrial processes, and CSCW systems merely provide assistance for groups of developers in collaborating and coordinating their activities.

This paper is an analytical review of a select set of the most prominent PSEEs introduced after the year 2003. Earlier PSEEs have been comprehensively reviewed in [20] and [28]. To ensure validity and effectiveness of the proposed evaluation criteria, PSEEs will first be described in terms of what they are supposed to provide (their requirements). The selected set of PSEEs will then be reviewed and evaluated based on a set of proposed criteria, specifically developed for this purpose as a part of this research; this critique, including an informative analysis of the evaluation results, will reveal their strengths and weaknesses.

The remainder of this paper is organized as follows: Section II characterizes the features that PSEEs are supposed to provide; Section III summarizes where we stand in PSEE Technology; Section IV reviews and evaluates the selected set of PSEEs, and extracts useful information from the evaluation results; and Section V presents the conclusions and suggests several directions for furthering this research.

II. CHARACTERIZING PSEES

In this section, PSEEs are characterized by the features that they provide, referred to as PSEE requirements. We will employ these requirements as the initial criteria for our evaluation framework in subsequent sections.

The conceptual and terminological framework shown in Fig. 1 was first introduced by Dowson [35] and has been used in several other related works. The framework introduces three software process domains:

- Process model domain
- Process enactment domain
- Process performance domain

The process model domain encompasses the definition and maintenance of software process models, which are defined in a PML. The process performance domain spans the set of actual tasks and activities that are performed by human or non-human process agents during the software process. Model and performance domains are linked together through the process enactment domain. The process enactment domain supports and controls the process performance domain through process models. In order to maintain the consistency and relevancy of software development process activities, it is essential for process enactment and process performance to be joined closely together. The close two-way communication between these domains is clearly depicted in Fig. 1.

Dowson's framework was employed in [37] and [33] to indicate the potential inconsistencies between the three conceptually distinguished software process domains, and to provide a solution to this problem. Reference [26] shows how process support takes place in each different domain. The methods presented in [15] and [20] have used the terminology of the framework to help present the key ideas and concepts of software process technology. This terminology will be used throughout the rest of this paper.

Another characterization of the software process domains is provided in Conradi's framework [36]. This framework divides a software process into meta-process, process support and production process. They are conceptually analogous to Dowson's framework's process model, process enactment and process performance domains respectively (as shown in Table 1). The research reported in [36] gives enlightening examples of the activities of each domain: modeling and instantiating the process from the process model domain, coding a method of a class during implementation from the production process domain, and a tool to monitor the actual state of process execution from the process support domain.

Conradi's framework is used in [32] to clarify the nature of PSEEs, as follows: A PSEE is supposed to provide not only meta-process support, but also a flexible mechanism to incorporate production process support. Activities like instantiating process models, monitoring the state of execution of the process, supporting the evolution of the process due to change requirements, allowing to associate agents with the process, and configuration management, provide meta-process support. Production process support is provided through a flexible integration tool and an interface to agents showing which activities to perform at which time.

A. PSEE Requirements

PSEEs have been characterized by many features, potentials and purposes. As mentioned earlier, software processes are considered as pieces of software, so we refer to these features as PSEE requirements. Based on Dowson's framework, some of the requirements of PSEEs are described below:

Enactment Support: The existence of a natural and simple way for implementing the process definition/instantiation/enactment paradigm is one important requirement of all PSEEs [32]. In order to support enactment, an execution environment based on well-defined PML(s) is required. Attaining a desired level of semi- or full automation of process activities is a possible outcome of process models enactment.

Software Team Distribution: The human-centered essence of software processes makes it necessary for PSEEs to support communication and coordination, not only between user performers, but also between users and automated process elements. Accordingly, the PML must include facilities to represent the effect of the performers' negotiation and interaction on the process state during process enactment. In order to ensure consistency between the states of the enactment and performance domains, performers must give the PSEE frequent feedback on the unexpected changes and decisions made about the software process, as illustrated in Dowson's framework (Fig. 1).

There are several works in the literature that discuss the importance of cooperation and coordination support in PSEEs. Referring to this feature as *groupware support*, reference [32] organizes the whole issue at two levels: the project level, and the organizational level: At the project level, a PSEE should support coordination – defined as workflow management between agents, and also collaboration – described as a kind of coordination in which agents need to share information; at the organizational level, the goal is to use the potential of PSEEs to build a persistent and reusable knowledge pool from the organizational experiences gained throughout the software process.

The research reported in [30] reviews the features and functionalities offered by SPADE, a well-known PSEE, to support cooperation. It remarks that specific cooperation policies are needed in the environment to effectively support integration with other tools and products. A more accurate boundary between coordination and cooperation is clarified by [16]: It introduces cooperative software engineering and defines cooperation as the manner of coordination in which an agreement on shared goals should be reached before development. In another work [17], collaboration is defined as users' awareness of each other's actions and coordination

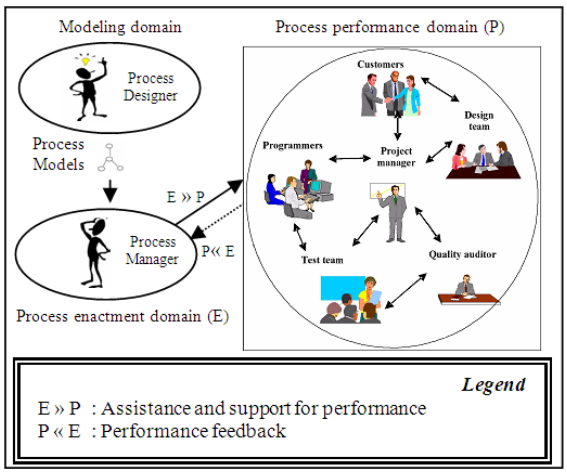

Figure 1. Software process domains [35]

is defined as mechanisms, like automatic distribution of work, aimed at avoiding excessive need for collaboration when supporting groups of software engineers.

Consistency management: PSEEs manage not only software processes, but also software products, which are subject to frequent change. There are direct as well as indirect dependencies between software products, so changes may pass from one product to another. In order to maintain consistency, PSEEs should properly handle these changes. Two common methods among PSEEs to prevent inconsistency have been presented in [25]: first, dependency relationships which make it possible to trace the chain of changes in software products; and second, preserving consistency by defining predicates that examine consistency conditions and trigger an exception in case of any consistency violation. One of these logic-based formalisms for managing consistency, which is based on modeling cooperative processes, has been proposed in [37].

Process flexibility: For our purpose, we define flexibility as the ability to modify software process activities dynamically during process performance. PSEEs should support this property to prevent the process enactment from diverging from the process performance [20]. From the viewpoint of the Dowson's framework in Fig. 1, E » P interaction is most effective only when PSEEs support software process flexibility. This way it is possible to modify and adapt the ordering of process actions during process performance, which is not the case if the PSEE supports only a fixed process model.

TABLE I. EQUIVALENT DOMAINS IN DOWSON'S AND CONRADI'S CONCEPTUAL FRAMEWORKS

Dowson's Framework Domain	**Conradi's Framework Domain**	**The Purpose of the Domain**
Process Model	Meta-Process	Process Modeling
Process Enactment	Process Support	Support and Control Production Activities
Process Performance	Production Process	Actual Production Activities

Process evolution: Like the E » P interaction, P » E feedback is also another essential feature of the Dowson's framework that should be supported by PSEEs. Software processes are performed by people, and human beings are not pieces of machinery. So the actual process which happens in the real world is subject to mistakes, oversights and deviations from the "observed process" [20], which is what the actual process is supposed to be. The observed process is also known as the "official process" [27]. Widening the gap between actual and observed processes is a potential consistency threat to process enactment and performance. It is only possible through performers' feedback to inform the PSEE of the process's evolution, and to meet this threat. PSEEs may also be used to improve the software process, which in its turn leads to software products improvement. Reference [32] suggests that PSESs are the right environments for addressing the CMM key process areas (KPAs) and facilitating process maturity improvement.

Modern Requirements: PSEEs should also be adapted to keep pace with significant changes in the world of software engineering and especially, the way software is developed. Reference [15] highlights agile software development and open source software development as the most significant challenges for PSEEs in terms of software process development. Dealing with agility and flexibility of agile methods in terms of process deviation is anticipated as the biggest challenge in developing "agile" PSEEs. Maintaining the flexibility of open-source software development while enforcing coordination processes has been pointed out as the most challenging difficulty in addressing open-source software development. Reference [22] also identifies the weakness of PSEEs in supporting issues such as security and rights to access shared artifacts as an obstacle in the adoption of open-source software development. Mobile software processes are another new challenge in the domain of PSEEs [21]. These processes include mobile process parts, and process participants with uncertainty about the place and prerequisites of their execution, mainly due to their mutable site allocation during the process.

A number of less important requirements that should typically be considered when implementing a PSEE have been discussed in [28].

III. STATE OF THE ART

A large number of PSEEs have been introduced in the literature, based on various concepts and approaches of software technology. This means that process programming has turned into a reality. Several reviews and classifications of current PSEEs have been reported in the literature. We will go through some of the most significant papers in this section, and will track the trend of technical viewpoint changes that have led to contemporary PSEEs. In the next section we will point out some weaknesses of current PSEEs.

Ambriola et al. [28] offer a thorough assessment of some contemporary PSEEs, based on a well-defined assessment grid. The grid is composed of certain critical points and challenging issues in the PSEE technology. All the assessment grid entries are organized in three main sections:

PML technology, PSEE architecture, and practical experiences of the PSEE. The assessment is then conducted based on the derived grid.

A more recent review on PSEEs has been reported in [20], in which a group of eight PSEEs have been selected and defined in terms of their objectives, PML features, and architectural characteristics. Based on a selected set of requirements for PSEEs, a comparative review of these PSEEs has been performed.

Further reports on PSEEs can be found in [21] and [23]. The former gives a brief overview of the key concepts and trends in the context of PSEEs. The latter is a more detailed version which presents a quick history of PSEEs and makes observations on future challenges in the field.

The research reported in [30] categorizes PSEEs into four groups, according to the support that they offer to their users: passive guidance, active guidance, process enforcement, and process automation. The difference between passive and active guidance is that in the first one, support is provided only upon user request, while in the second one, the PSEE may ask for user intervention on-demand. In process enforcement, the user is asked to perform activities specified by the PSEE, whereas in process automation, tasks are performed without involving the user.

The research reported in [27] gives two other classification criteria for PSEEs. The first one is to classify them into proactive and reactive groups. Control and initiation of the operations is performed by the environment and by the user in each group respectively. The second one is to arrange PSEEs into four groups with respect to the paradigm used in the corresponding PML. One approach is to extend conventional programming languages and adapt them to the process programming needs. Another approach is to use rule-based languages to define the software process. Using preconditions, actions, and postconditions of rules, it is possible to describe software process activities. State machines are another powerful means for modeling the software process. So in the third approach, automata-based formalisms like Petri nets are extended to provide a practical solution to software process modeling. Finally, it is also possible to blend two or more distinct techniques and utilize the combination to benefit from each technique's advantages; this is called the multi-paradigm approach.

Recently, the collaborative qualities of PSEEs are being highlighted and emphasized more than other attributes [16, 17]. This trend depicts a more promising future for PSEEs, based on the fact that approximately 70% of the software developers' time is consumed for collaborative tasks. The research reported in [12] presents a classification framework that organizes the different research areas on collaborative software development.

Collaborative software development, when geographically distributed, forms the paradigm of federated PSEEs. The research reported in [31] provides a brief introduction to the goals and motivations of federated PSEEs. It also makes a list of open issues of the technology. Efforts aimed at addressing these problems have already been started; the research reported in [10] proposes a peer-to-peer solution instead of the old client-server architectures for cooperative development of software processes in highly-dynamic environments.

Other works have focused on improving PSEE technology by using new software concepts. For instance, the research reported in [11] blames traditional software concepts applied to PSEEs for the limited success of PSEEs in the software industry. Agent-based PSEEs is the idea proposed by the authors. Intelligence, autonomy and the reasoning abilities of agents are utilized to manage software development activities, and to act in different software process phases, just as software developers do. Groups of process agents, called process agent profiles, cooperate with each other under a framework called the Management Net to conduct process enactment.

A. *A critique of current PSEEs*

With serious rival technologies in the software process industry, it seems that reconsiderations should be made as to the scope and purpose of PSEEs. In this section, we will examine some of the most serious weaknesses of current PSEEs.

The most severe problems of PSEEs seem to be more about their practicality rather than their practicability. In other words, due to their complexity and inflexibility (that is, their *intrusiveness* [24]), PSEEs are ignored or abandoned by process engineers during process execution. Flexibility, defined as tolerance for inconsistencies and process deviations – mentioned earlier as one of the requirements of PSEEs – is essential for these environments. To inject flexibility into a PSEE's design and resolve the problem of strictness of process support in PSEEs, several mechanisms have been proposed in [21]; examples include: concise activity description at early phases of development, and guidance offered by the environment upon user request. However, flexible handling of deviations during software process enactment is still a major research area [2, 3].

PSEEs also fail in human-related and creativity-involved aspects of software development. As pointed out in [21], software processes include parts that demand the creativity of human minds; this should also be supported by PSEEs along with other process parts. In order to effectively fulfill human-related needs, PSEEs should contain flexible mechanisms to support late changing requirements, emerging and evolving technologies, and dynamic working environments. Since human dimension support is a new field of development in PSEE technology, it is open to further research and enhancements.

In order to avoid making the same mistakes in the design and development of future PSEEs, an enlightening comparison has been reported in [27]. It makes a list of some strategic decision points that should be considered when designing a PSEE, defines extremes at each decision point, and compares their technical effect on the PSEE.

IV. EVALUATION OF SELECTED PSEES

In this section, we have presented a review of the significant aspects and distinguishing features of a set of seven PSEEs which were introduced after 2003. As noted before, earlier PSEEs have been comprehensively reviewed

in [20] and [28]. Each PSEE is briefly characterized in terms of its applications (and thereby its prominent contributions), as well as through an informative architectural diagram. A comprehensive criteria-based evaluation is then performed on the examined PSEEs, based on a criterion set that has been developed as an important contribution of this research. The evaluation results are then analyzed and discussed to finalize the evaluation.

A. DOSDE Environment

DOSDE (Domain-Oriented Software Development Environment) is the name of a PSEE which has been based on the concept of employing domain knowledge during the software process in order to support development activities [19]. Domain-specific knowledge is the essential prerequisite of the critical task of properly identifying software requirements. In DOSDE, this knowledge is divided into: 1.the knowledge about the concepts of the application domain, and 2.the knowledge about the typical tasks and activities of the domain. Considering the need for generality and reusability of the definition, this knowledge is captured in the form of interrelated ontologies.

In order to cope with the complexity of the application domain ontology, and also to facilitate its definition, the ontology is divided into sub-ontologies. A sub-ontology is a group of related domain concepts which share the same semantic context. Application domain sub-ontologies are constructed through the following phases: definition of the purpose of the ontology, conceptualization, formalization (ontology coding), and validation. On the other hand, task ontologies, augmented by problem solving methods (PSMs), aim at deepening the understanding of the users of the domain tasks. While a task ontology merely conceptualizes the corresponding task, one or more PSMs are associated with the task to solve it.

The mentioned concepts are utilized in DOSDE environment in assisting requirements elicitation and documentation, and also the design activity. Use cases are described in the form of task ontology and associated PSMs. During design, data modeling is greatly facilitated in DOSDE owing to a mapping definition between ontological constructs and entity relationship diagrams. In addition, DOSDE studies and stores domain-specific information by extending various activities of the development process with a sub-activity called domain investigation activity. Fig. 2 provides a visual representation of DOSDE's underlying concepts. Please note that the same legend applies to all the subsequent figures.

B. VRPML Support Environment

A PML, called VRPML (Virtual Reality Process Modeling Language) and the design of its support environment are discussed in [13]. In VRPML, software processes are generically specified as graphs. These graphs are constructed from interlinked nodes, representing the process activities, and arcs, indicating the control-flow of the process. Different types of activity nodes are supported in VPRML, including: general-purpose activity nodes, multi-instance activity nodes, and meeting activity nodes. A start node generates the initial control-flow signal, which subsequently triggers activity nodes from top to bottom of the graph. Decomposable transitions also enable conditional branching of process control-flow. Two VPRML elements, named merger and replicator, make it possible to concurrently enact activity nodes. The language also provides a macro node, which is a packaging mechanism to enhance the readability of the graph.

The main components of the VPRML support environment architecture, illustrated in Fig. 3, are briefly discussed herein: Graph editor is a dedicated visual editor to draw and modify the VPRML graphs. The graphs are then translated into a ready-to-enact intermediate format by compiler, named roadmap. Run-time interpreter not only parses and interprets the roadmap, but also decides when to fire activities on arrival of the control signal and interacts with the resource manager to check for resource availability. Runtime client retrieves activities and their resource allocations from communication repository layer. To-do-list manager has responsibility for managing the assigned activities to a particular software engineer. Each activity work context, including artifacts and tools required to complete the activity, is generated and maintained by the workspace manager. Communication repository layer is a intermediate container for keeping assigned activities and the control-flow signal. Finally, resource manager handles the queries for resource allocation received from the runtime interpreter and the workspace manager.

C. CASDE Environment

In order to effectively support software development processes in general, and collaborative development activity in particular, the architecture of a context-aware software development environment, or CASDE, has been presented in [9]. As inferred from the name, the context element has the pivotal role in the proposed architecture. The context of an entity, that contains state- and location-specific information about the entity, makes it possible to share artifacts between different participants. The key aim of CASDE is to support collaborative features in software development.

CASDE is theoretically founded on the activity theory; it thus supports collaborative activities in a three level hierarchical structure, as identified by the activity theory: co-ordinated level, co-operative level, and co-constructive level. The co-ordinated level deals with routine flows of interactions, and accentuates the individuals performing their assigned activities and roles. The co-operative level focuses on the problem of actors sharing a common object and cooperating to achieve a shared objective. The context element plays an integral part in providing synchronous communication at the co-operative level. The co-constructive level is placed at the top of the collaborative activity hierarchy and involves interactions that concern re-conceptualization and gradual evolution of the process.

The CASDE architecture is presented in Fig. 4. The architecture follows the client-server model, and consists of interconnecting modules at client-side as well as server-side.

Two distinctive modules of CASDE architecture are wrapped legacy tools and awareness tools. Legacy tools can

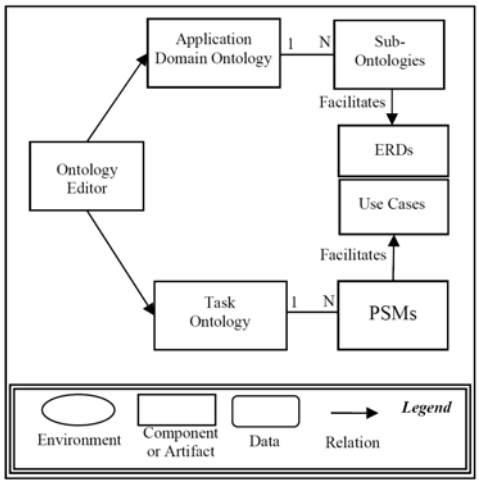

Figure 2. DOSDE concepts

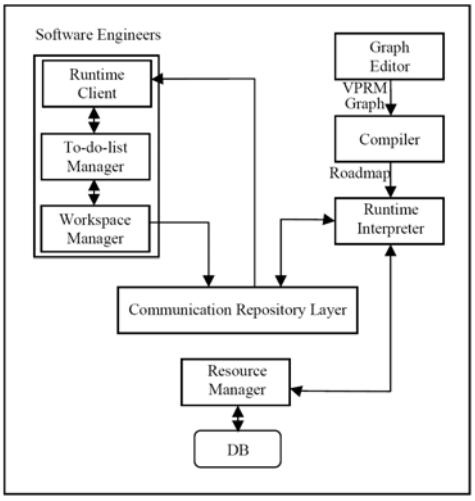

Figure 3. VRPML Architecture (adapted from [13])

be easily adopted in CASDE owing to the tools wrapping mechanism. The local repository at client-side maintains local versions of artifacts. Awareness tools update artifact contexts and notify the developers of the changes.

D. Transforms Environment

According to its developers, Transforms is the first PSEE dedicated to modeling and enactment of MDA processes, i.e. model-driven software development processes which adopt Model Driven Architecture concepts and employ model transformation tools for building software-intensive systems [6]. Since MDA specification does not, define a software process [18], one main goal of the underlying concept of Transforms is to fill the gap between MDA standards and a clear-cut process definition for applying MDA concepts. Unlike most MDA tools and environments, Transforms focuses on all process design aspects during enactment, rather than just on model transformations.

We have already mentioned SPEM as a standard notation for process modeling. Based on a specialized set of SPEM 2.0 concepts, Transforms offers: (1) a meta-model for defining and instantiating an MDA process extended from certain SPEM concepts, and (2) a set of diagrams for modeling the behavior and structure of the MDA process. The incorporation of SPEM encourages interoperability and interchangeability of models between model transformation tools, which is the major weak point of most other MDA environments. In Transforms, MDA-related particulars are preferred to be defined at the metamodel layer (M2) instead of the process model layer (M1) [6]; this way, distinctive MDA process definitions (encompassing different features) can be modeled in M1 and then enacted and used at the process development layer (M0).

Fig. 5 shows the architecture of the Transforms environment. The MDA process modeling module provides a visual editor to modify the process's structure and behavior, and rule and profile editors to define new transformation rules and UML profiles. On the other hand, model transformations and the ability to monitor tasks and artifacts are supported by MDA process enactment module.

E. SPACE Environment

Mainly targeted at medium-sized projects and small software teams, SPACE (Semantic Process- and Artifact-Oriented Collaboration Environment) is a conceptual meta-model for creating and managing process models as well as artifact models, and applying the models to support the core and context activities of the software development process [5]. Core and context activities are respectively oriented towards the creation of the actual product, and project planning and management. Artifact models are semantically associated with process models to define the artifact transformation chain throughout process execution, and also to incorporate traceability into the process.

In order to tailor SPACE to the needs of small organizations, dynamic and flexible pre-defined process models should be defined and applied to the development processes of these companies. Since SPACE is a domain-independent meta-model, it has been adapted and applied to software engineering based on a software platform called SOP (Software Organization Platform), which encompasses three fundamental components: *Lifecycle Artifact and Process Management, Knowledge Management, and Stakeholder Collaboration.* SOP 2.0 is a user-driven wiki-based platform which implements SOP. The relationship between SPACE, SOP and SOP 2.0 is depicted in Fig. 6.

F. The ADAMS System

Advanced Artifact Management System, or ADAMS, is an artifact-based environment that integrates project and artifact management features to support software engineering activities during the entire software process lifecycle [14, 4]. Instead of adopting the common approach of process definition/instantiation/enactment, ADAMS makes it possible to define a project in terms of the artifacts to be produced and the relationships among them. Configuration management and traceability support for artifacts in the early phases of software lifecycle, and fine-grained management of artifacts which enables control and traceability of

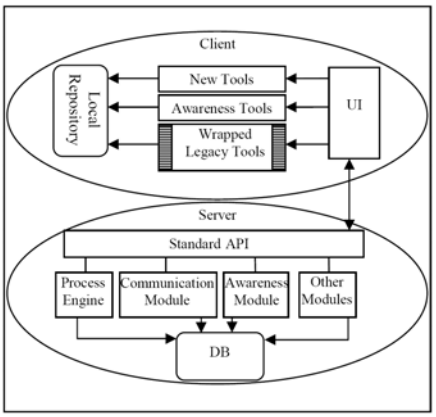

Figure 4. CASDE Architecture (adapted from [9])

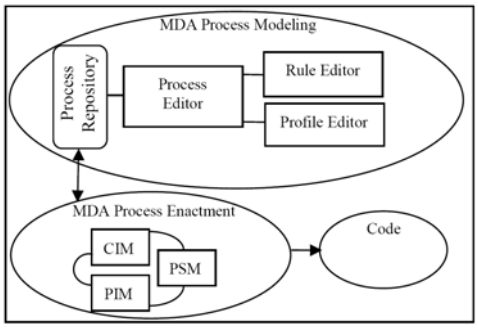

Figure 5. Transforms Architecture [6]

independent elements in a file, are some of the distinct advantages of ADAMS. ADAMS architecture, as show in Fig. 7 is composed of several subsystems; each subsystem has clearly-defined responsibilities, as follows: The resource management subsystem implements a role-based access control policy to provide administrative functionalities and account management. The project management subsystem is responsible for providing the management-related functionalities around the pivotal role of artifacts. Tasks and activities are represented by the artifacts that they are supposed to create as their output. The project schedule is also based on the start and due date of the artifacts. Resources are assigned to artifacts, and specific roles are assigned to resources according to their roles in developing each artifact. The artifact management subsystem provides fine-grained management and revision control of software artifacts. The event and notification management subsystem enhances traceability of artifacts through providing context-awareness within the project. The quality management subsystem guarantees the quality of artifacts using related software engineering practices, such as: employing standard templates, providing review checklists for artifacts, and software inspection to identify defects and reduce rework. Finally, synchronous and asynchronous collaborative tools, such as internal chat, collaborative UML editors and internal email, form the cooperative development subsystem.

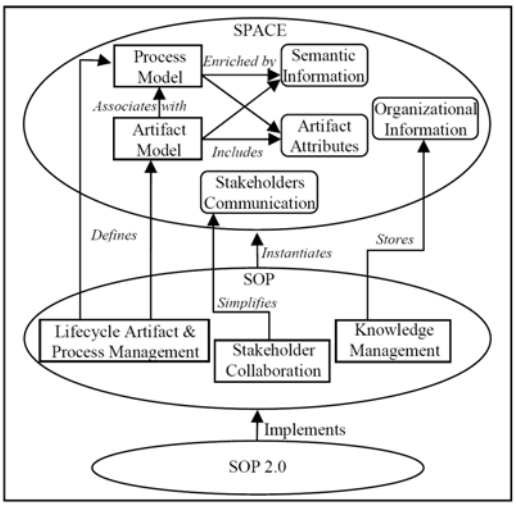

Figure 6. Genealogy of SPACE and SOP Platforms

G. *Model-Driven Integrated approach*

The approach introduced in [1] integrates the principles and techniques of model-driven engineering and software product lines to adopt a model-driven integrated approach to software process definition, customization and execution. The approach is implemented through using several well-known software technologies; since it can be considered as a PSEE (at least potentially), it has been selected herein.

Fig. 8 gives an overall understanding of the main elements of the Model-Driven Integrated approach and their respective relationships. As the first step, modeling and definition of the software process model takes place, relying on supporting EPF technologies [39], including the Unified Method Architecture meta-model, EPF Composer tool, and the method content; i.e., the process assets to be used as the basis for composing a broad range of processes. The next step concentrates on process variability management, which is perhaps the most prominent advantage of this approach. Here, a variability model is defined to specify existing variabilities of a software process. The variability model includes EMF models that specify which process fragments (roles, tasks, etc.) represent variabilities (optional and alternative) when regarding specific projects. Customized versions of the software process are then derived from an existing software process. This is done automatically by the GenArch product derivation tool through selection of the relevant features from an existing process. The two final steps are straightforward: First, a model-to-model transformation language called ATL, an implementation of OMG Query/View/Transformation language, is used to translate EPF specification to JPDL workflow specifications; finally, this JPDL specification is converted to Java Server Faces (JSF) by a model-to-text language named Acceleo, which can be deployed and executed by the JBoss Business Process Management (jBPM) workflow engine.

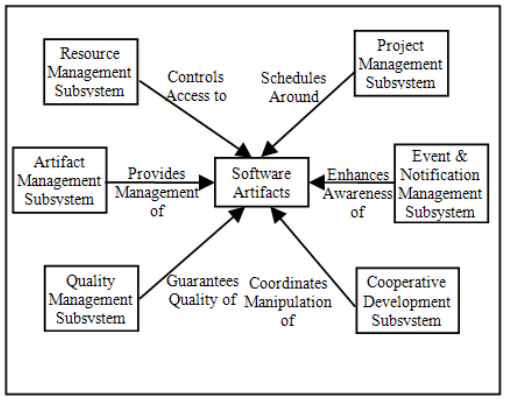

Figure 7. ADAMS Architectue

H. Evaluation of the selected PSEEs

Herein, we present a detailed criteria-based evaluation of the selected PSEEs using an evaluation method adopted from the Feature Analysis approach [29]. In our approach, each evaluation criterion is further divided into several more features. Each feature is given a weight, identifying the value of the feature in comparison to other equivalent features. Weight is a floating number ranging from 0.0 to 1.0, and the sum of the weights of all the features related to one criterion should equal 1.0. As in the Feature Analysis approach, we differentiate between narrative and scale features. A feature belongs to the narrative type if a PSEE can be simply evaluated against that feature by a "0" or "1" response. A "1" response means that the PSEE explicitly incorporates the corresponding feature; while a "0" response means the feature is not present in the PSEE. For instance, support for traceability between artifacts is a narrative feature in table III with a weight of 0.5, presented in the "Weight" column.

Evaluating a PSEE against some other features, on the other hand, may necessitate defining a scale to accurately evaluate the compliance degree of the PSEE to that scale feature. Process definition support in table II is an example of scale features which cannot be simply evaluated by a "yes/no" response. Symbols *N* and *S* in the "Type" column, respectively represent narrative and scale features in tables II, III and IV. In addition, an extra index for scale features associates one of the following scales with that feature:

Scale 1 (S_1). Scale 1 includes 4 levels: **0/3**: the PSEE doesn't mention the corresponding feature and thus, provides no mechanism to support it; **1/3**: the PSEE's definition merely highlights the corresponding feature importance and/or provides guidelines/extension points to extend the PSEE functionality to incorporate the feature; **2/3**: the PSEE partially supports the corresponding feature via other key modules of the architecture, but no specific mechanism aimed at the desired feature is defined in the PSEE definition; **3/3**: the PSEE totally supports the corresponding feature and accentuates the mechanism(s) supporting the feature, as a dedicated module of the PSEE architecture.

Scale 2 (S_2). Scale 2 includes 3 levels: **0/2**: the PSEE doesn't explicitly cover any part of the corresponding feature or paradigm; **1/2**: the PSEE partially covers some parts of the corresponding feature or paradigm; **2/2**: the PSEE covers and includes all parts of the corresponding feature or paradigm.

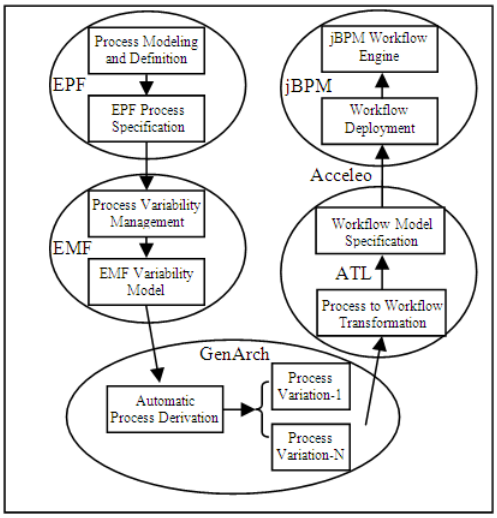

Figure 8. The Model-Driven Integrated Approach overview (adapted from [1])

We hereby provide examples of employing the defined scales in order to evaluate the reviewed PSEEs against scale parameters: As seen in table V, the Transforms environment is evaluated based on the process flexibility criterion, resulting in a $2/3$ mark. This is because flexibility is only partially provided in Transforms through its meta-model modification. As the second example, CASDE is given a $1/3$ mark evaluating against process evolution criterion, since process evolution is only mentioned at co-constructive level of CASDE, which is not currently supported by CASDE. It should be noted that both process flexibility and process evolution support criteria are composed of only one feature, as shown in table II. If the criterion is comprised of more than one feature, the mark presented in the table entry is the sum of the values resulting from evaluating the PSEE against all the corresponding features. For example, consider evaluating SPACE against the coverage criterion. Coverage consists of two features associated with scale 2, as defined in table IV. SPACE is given a 2/2 mark and a 1/2 mark for the first and second features, respectively. Eventually, the corresponding weights are multiplied by these marks (2/2*2/3+1/2*1/3), resulting in the table entry (5/6).

The initial set of evaluation criteria have been derived from PSEE requirements and the results of the critique provided in the previous section. This strategy guarantees the validity and effectiveness of the resulting evaluation criteria. Accordingly, the extracted criteria have been organized into three groups: 1.Evaluation criteria derived from PSEE requirements, 2.Evaluation criteria derived from the critique results, and 3.General software process evaluation criteria.

The final criterion set used here to evaluate PSEEs is evolved and refined from the initial set. During the evaluation, the count, granularity, type, and weight of the criteria and features have been subject to refinement. For example, interoperability was added as a criterion to general evaluation criterion set, and tool support was added as a

feature to the process deviation support criterion, after the evaluation of Transforms environment. While these features are crucial to Transforms as a PSEE adopted for MDA processes, they are also essential regarding a typical PSEE. As another example, traceability, which can be efficiently supported by the automation offered by a PSEE, was added to general evaluation criteria after the evaluation of SPACE. The scaling rules have also been evolved as one target of the evolution process. For example, scale 2 was wholly added to fit the evaluation of the coverage criterion. We have also selected and adapted certain general evaluation criteria, such as usability, from [7], which provides an analytical review of Computer Aided Method Engineering (CAME) tools.

During the criteria definition and refinement process, we have strived to keep the criteria general, precise, comprehensive and balanced in order to satisfy the meta-criteria of [34]. The final criteria, categorized in the three mentioned groups, are shown in tables II, III and IV.

This review aims to address the need for a review on modern PSEEs. The reviews reported in [20] and [28] were conducted a decade ago; the PSEEs reviewed here had not been introduced at that time. Using the matrix framework, employing scaled features, and introducing evaluation criteria that are related to modern software technologies, are the distinguishing features of the present research.

TABLE II. EVALUATION CRITERIA DERIVED FROM PSEE REQUIREMENTS

Criterion Name	Corresponding Feature	Type	Weight
Enactment Support	Process definition support	S_1	1/3
	Process instantiation support	S_1	1/3
	Process execution support	S_1	1/3
Software Team Distribution	Coordination (Workflow Management) Support	S_1	1/2
	Collaboration (Information Sharing) Support	S_1	1/2
Consistency Management	Consistency Preservation Mechanism Support	S_1	1/1
Process Flexibility	Dynamic Process Activities Modification Support	S_1	1/1
Process Evolution	Process Performer's Feedback Mechanism Support	S_1	1/1
Security	Role-Based Access Control	N	1/1
Mobility	Mobility of Process Parts	S_1	1/1

TABLE III. EVALUATION CRITERIA DERIVED FROM THE RESULTS OF THE CRITIQUE

Criterion Name	Corresponding Feature	Type	Weight
Process Deviations Support & Inconsistencies Toleration	Concise activity description at early development phases support	N	1/5
	Monitoring of process activities support	S_1	2/5
	Offering guidance and support by environment upon request support	N	1/5
	Tool Support for Handling Process Deviations	N	1/5
Human Dimension Support	Late Changing Requirements support	N	1/4
	Process experience knowledge pool support	S_1	1/4
	Usability	S_1	2/4
New Technology Adoption	Adopting Modern Software Technologies in PSEEs	N	2/3
	Introducing Conventional Software Concepts to PSEEs	N	1/3

TABLE IV. GENERAL SOFTWARE PROCESS EVALUATION CRITERIA

Criterion Name	Corresponding Feature	Type	Weight
Coverage	Supporting Whole Generic Life Cycle	S_2	2/3
	Supporting Umbrella Activities	S_2	1/3
Change Management	Configuration Management	N	1/2
	Quality Management	S_2	1/2
Traceability	Traceability Between Artifacts Support	N	1/2
	Traceability Between Roles and Activities Support	N	1/2
Interoperability	Using a Standard PML	N	2/3
	Extension Points Support for Integrating Tools	N	1/3

I. Analysis of the Evaluation Results

Table V summarizes the results of applying the evaluation criteria to the reviewed PSEEs. The results can be argued and used for two distinct purposes: First, they can be used to compare the evaluated PSSEs, and to select an appropriate PSEE in a specific usage context based on a set of predefined requirements. Second, the evaluation results can be used alongside the findings from the review to identify recent trends in PSEE technology.

Normally, most of the PSEEs succeed in meeting the conventional PSEE requirements to an acceptable extent. Transforms and MD-Integrated are the only exceptions, in that they fail to meet some of the most trivial requirements of PSEEs. This is mainly due to the fact that they are still incomplete. On the other hand, modern PSEE requirements, namely security and mobility, are not addressed in most of the current PSEEs. Also, PSEEs still fail in some of the general evaluation criteria and the criteria derived from the results of the critique, just as their predecessors did.

One growing trend among modern PSEEs, such as SPACE, Transforms and the MD-Integrated, is the use of a metamodel to define the process model and also to instantiate and execute the process instance from the process model. In this manner, they effectively support process enactment through the employed definition/instantiation/enactment paradigm.

ADAMS, unlike other reviewed PSEEs, adopts an artifact-oriented approach to software process and does not comply with the conventional paradigm of software process definition/instantiation/enactment. Transforms and MD-Integrated, which employ some of the more modern software standards and technologies – such as SPEM, model-driven development and EPF technologies – succeed in meeting some of the more challenging criteria, such as process deviation support and interoperability.

V. CONCLUSIONS AND FUTURE WORK

In this paper, a brief but comprehensive survey on the current state of process-centered software engineering environments was presented. We went through PSEE requirements and demonstrated their capabilities and shortcomings. A short critique was then conducted in the hope of tackling the problem of limited acceptance of PSEEs in the software industry. The results of this critique can be employed to construct and examine new PSEEs in order to avoid the pitfalls. We then selected a set of seven modern

TABLE V. EVALUATION OF THE SELECTED PSEES

Criteria Group	Criterion Name \ PSEE Name	DOSDE	VRPML SE	CASDE	Transforms	SPACE	ADAMS	MD Integrated
PSEEs' Requirements	Enactment Support	3/3	3/3	2/3	3/3	3/3	3/3	3/3
	Software Team Distribution	1/2	2/2	2/2	0/2	2/2	2/2	0/2
	Consistency Management	0/3	2/3	2/3	0/3	3/3	3/3	0/3
	Process Flexibility	0/3	1/3	3/3	2/3	3/3	3/3	3/3
	Process Evolution	0/3	3/3	1/3	3/3	0/3	3/3	0/3
	Security	0/1	1/1	0/1	0/1	1/1	1/1	0/1
	Mobility	0/3	0/3	0/3	0/3	0/3	0/3	0/3
Critique Results	Process Deviations Support	12/30	6/30	18/30	21/30	2/30	3/30	12/30
	Human-Dimension Support	12/24	2/24	5/24	12/24	18/24	18/24	12/24
	New Technology Adoption	3/3	0/3	1/3	2/3	2/3	0/3	2/3
General Requirements	Coverage	2/6	0/6	1/6	3/6	5/6	6/6	4/6
	Change Management	2/4	1/4	1/4	3/4	1/4	4/4	2/4
	Traceability	0/2	0/2	0/2	1/2	2/2	2/2	0/2
	Interoperability	0/6	0/6	2/6	5/6	4/6	2/6	6/6

PSEEs and provided a detailed review and criteria-based analysis of each PSEE. We aim to further this research by extending our evaluation framework and developing a generic PSEE framework that can be instantiated to yield custom PSEEs.

REFERENCES

[1] F.A. Aleixo, M.A. Freire, W.C. dos Santos, and U. Kulesza, "Automating Variabitliy Management, Customization and Deployment of Software Processes: A Model-Driven Approach", pp. 372–387, Springer, 2011.

[2] M.A. Almeida da Silva, R. Bendraou, J. Robin, X. Blanc, "Flexible Deviation Handling during Sofware Process Enactment", 15th IEEE International Enterprise Distributed Object Computing Conference Workshops, 2011.

[3] M.A. Almeida da Silva, R. Bendraou, X. Blanc, M.P. Gervais, "Early Deviation Detection in Modeling Activities of MDE Processes", LNCS, vol. 6395, pp. 303-317, Springer, 2010.

[4] A.D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Fine-grained management of software artefacts: the ADAMS system", Software: Practice and Experience, 40(11):1007–1034, 2010.

[5] S. Weber, A. Emrich, J. Brocher, E. Ras, and Ö. Ünalan, "Supporting Software Development Teams with a Semantic Process- and Artifact-oriented Collaboration Environment", in Workshop on Collaboration and Knowledge Sharing in Software Development Teams (SOFTEAM2009 at Software Engineering Conference), Kaiserslautern, Germany, 2009.

[6] R. S. P. Maciel, B. C. Silva, and N. S. Rosa, "An Integrated Approach for Model Driven Process Modeling and Enactment", in XXIII Brazilian Symposium on Software Engineering, 2009.

[7] A. Niknafs, R. Ramsin, "Computer-Aided Method Engineering: An Analysis of Existing Environemnts", LNCS, vol. 5074, pp. 525–540, Springer, 2008.

[8] OMG. "Software Process Engineering Meta-model Specification", Version 2.0, (formal/08-04-01) (2008).

[9] T. Jiang, J. Ying, M. Wu, M. Fang, "An Architecture of Process-centered Context-aware Software Development Environement", Proceedings of the 10th International Conference on Computer Supported Cooperative Work in Design, IEEE, 2006.

[10] D. Balzarotti, C. Ghezzi, M. Monga, "Supporting Cooperative Software Processes in a Decentralized and Nomadic World", IEEE Transactions On Systems, Man, And Cybernetics—Part A: Systems And Humans, VOL. 36, No. 6, November 2006.

[11] X. Zhao, K. Chan, M. Li, "Applying Agent Technology to Software Process Modeling and Process-Centered Software Engineering Environments", ACM Symposium on Applied Computing, 2005.

[12] A. Sarma, "A Survey of Collaborative Tools in Software Development", ISR Technical Report, UCI-ISR-05-3,March, 2005.

[13] K. Z. Zamli, N. A. Mat Isa, N. Khamis, "The Design And Implementation Of The VRPML Support Environments", Malaysian Journal of Computer Science, Vol. 18 No. 1, June 2005, pp. 57-69.

[14] A.D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, "ADAMS: An artefact-based process support system", Proceedings of the Seventh International Conference on Software Engineering and Knowledge Engineering, Canada, 31-36, 2004.

[15] J. C. Derniame, and F. Oquendo, "Key Issues and New Challenges in Software Process Technology", the European journal for the informatics professional, Vol. V, NO. 5, October 2004.

[16] L. Liang, Y. Tang, W. Xiao, A. Feng, "The Literature Review of Cooperative Software Engineering", Proceedings of the 8th international conference on computer supported cooperative work in design, 648- 652, May 2004.

[17] P. Barthelmess, "Collaboration and Coordination in Process-Centered Software Development Environments: a Review of the Literature", Information and Software Technology, 45 (2003) 911–928, 2003.

[18] I. Mukerji, and J. Miller, MDA Guide Version 1.0.1. OMG, 2003.

[19] K. M. de Oliveira, F. Zlot, A R. Rocha, G.H. Travassos, C. Galotta, and C.S. de Menezess, "Domain-oriented software development environment", Journal of Systems and Software, 2003.

[20] S. ARBAOUI, J. C. DERNIAME, F. OQUENDO, H. VERJUS, "A Comparative Review of Process-Centered Software Engineering Environments", Annals of Software Engineering, 14, 311–340, 2002.

[21] V. Gruhn, "Process-Centered Software Engineering Environments: A Brief History and Future Challenges". Annals of Software Engineering, 14, 363–382, 2002.

[22] K. Z. Zamli, "Process Modeling Languages: A Literature Review", Malaysian Journal of Computer Science, Vol. 14, No. 2, 2001.

[23] G. Engels, W. Schafer, R. Balzer, V. Gruhn, "Process-Centered Software Engineering Environments: Academic and Industrial Perspectives". Proceedings of ICSE'01, 671 - 673 2001.

[24] A. Fuggetta, "Software Process: A Roadmap", The Future of Software Engineering (FOSE 2000) in Conjunction with ICSE 2000, Limerick, Ireland. ACM Press, June 2000.

[25] J. J. Chen, S. C. Chou, "Consistency Management in a Process Environment", The Journal of Systems and Software, 105-110, 1999.

[26] K. Pohl, K. Weidenhaupt, R. Domges, P. Haumer, M. Jark, R. Klamma, "PRIME: Towards Process-Integrated Environments", ACM Transactions on Software Engineering and Methodology, 1999.

[27] G. Cugola, A. Fugetta, "Software Processes: a Retrospective and a Path to the Future", Software Processes: Improvement and Practice, 4(3), 100-104, 1998.

[28] V. Ambriola, R. Conradii, A. Fuggetta, "Assessing Process-Centered Software Engineering Environments", ACM Transactions on Software Engineering and Methodology, 6(3), 283-328, 1997.

[29] B. Kitchenham, S. Linkman, D. Law, "DESMET: a methodology for evaluating software engineering methods and tools". Computing and Contrological Engineering Journal~8, 120-126 (1997).

[30] S. Bandinelli, E. Di Nitto, A. Fuggetta, “Supporting Cooperation in the SPADE-1 Environment". IEEE Transactions on Software Engineering, 22(2), December 1996.

[31] C. Basile, S. calanna, E. Di Nitto, A. Fuggetta, M. Gemo, “Mechanisms and policies for federated PSEEs: basic concepts and open issues”, Proceedings of the 5th European Workshop on Software Process Technology, LNCS, 1996.

[32] E. Elmer, “Process-Centered Software Engineering Environments as the Next Generation of CASE Tools”, Next Generation Case Tools workshop 95, Feb. 1995.

[33] B. Nuseibeh, “Computer-Aided Inconsistency Management in Software Development”, Technical report DoC, 1995.

[34] G.M. Karam, R.S. Casselman,“A cataloging framework for software development methods”. IEEE Computer~26(2), 34—45, 1993.

[35] M. Dowson, “Consistency Maintenance in Process Sensitive Environments”. In Proceedings of the Process Sensitive SEE Architectures Workshop, 1992.

[36] R. Conradi, C. Fernström, A. Fuggetta, R. Snowdon, “Towards a Reference Framework for Process Concepts”, Proceedings of EWSPT-2, LNCS 635, Springer, pp. 3-17, 1992.

[37] I. Alloui, F. Oquendo, “Managing Consistency in Cooperating Software Processes”, Lecture Notes in Computer Science, Volume 1487/1998, 92-99, 1992.

[38] L. Osterweil, “Software Processes Are Software too”, In Proceedings of ICSE’87, ACM Press, 2-13, 1987.

[39] Eclipse Process Framework - http://www.eclipse.org/epf/.