

Concepts for Evolving Software Processes

Reidar Conradi*, Norwegian Institute of Technology, Norway
Christer Fernström†, CAP Gemini Innovation, France
Alfonso Fuggetta‡, Politecnico di Milano and CEFRIEL, Italy

Submitted for PROMOTER book, V.14

December 2, 1993

Abstract

Software processes are complex entities that may last for long periods of time and are carried out through the interaction of humans and computerised tools. They need to continuously evolve in order to cope with different kinds of changes or customisations both in the organisation and in the technologies used to support software production activities.

In recent years, many software process technologies have been developed, and have currently been further extended and used in trial projects. Moreover, some research prototypes have generated commercial products, that are marketed and currently used in industrial organisations. Despite these significant efforts and results, however, there is still little conceptual characterisation and assessment of the properties of software processes and related support environments. It is difficult to compare and assess existing approaches. Even a common characterisation of the problems to be addressed seems to be problematic and difficult to achieve. This is particularly true when we consider software process evolution, for which it seems that an agreed-upon and satisfactory solution has not been established yet.

*Dept. of Computer Systems and Telematics, NTH, N-7034 Trondheim, Norway. Tel.: +47-73-593444, Fax: +47-73-594466, E-Mail: conradi@idt.unit.no.

†CAP Gemini Innovation, 7 Chemin du Vieux Chêne, F-38240 Meylan, France. Tel.: +33-76-764720, Fax: +33-76-764748, E-Mail: christer@capsogeti.fr.

‡Politecnico di Milano, Dipartimento di Elettronica e Informazione, Piazza Leonardo da Vinci 32, I-20133 Milano, Italy. Tel.: +39-2-239.93623, Fax: +39-2-239.93411, E-Mail: Alfonso.Fuggetta@mailier.cefriel.it.

This work aims at proposing a conceptual framework to describe and assess flexible and evolving (software) processes. It is based on the assumption that a process is composed of two main components: a *production process* to carry out production activities, and a *meta-process* to improve and evolve the whole process.

The general requirements and properties of the (software) process domain are first discussed, and the meta-process concept is introduced. Then, we discuss several process-related concepts and, in particular, the relationship between the meta-process and the rest of the process. Methods and technologies needed to support the meta-process are highlighted and discussed. Finally, we apply the resulting framework to an example, in order to show the potential and expected benefits of the proposed approach.

Keywords: (software) process, process modelling, process evolution and improvement, meta-process.

1 Introduction

During the last decades the problem of producing high quality software products has become increasingly complex and difficult to manage. One reason for this is the rapid evolution of technologies and methods to produce software, together with an increased complexity of the applications to be developed. These two factors are strongly interrelated: advances in the technology enable creation of new products, services and activities, or modification of old ones, which in their turn produce new needs, feedbacks, and requirements to software technology providers. Further, software processes are *human-oriented* [CFFS92], and the interactions among humans and between humans and the tools, that support their activities are characterised by high variability and unpredictability. This fact further increases the complexity of the resulting software process, and puts hard demands on management. Finally, software processes may last for long periods of time, and are thus likely to undergo many changes during their lifetime in order to cope with new requirements [BL79]. Examples of such changes are the substitution of (part of) the technologies used to produce software applications, or the modification of the development strategy and procedures. Other important reasons for software process evolution is the need to (dynamically) customise the software process to accommodate requirements or preferences of individuals that take part in the process, or to cope with various unanticipated situations.

The productivity of software providers and the quality of the provided software products and services intrinsically depend on the effectiveness of the associated process. This insight has steered work on software quality and productivity towards improvement of software processes:

- How can a software process be described, in order to understand it and to facilitate communication and teaching?
- How can the software process be managed and improved?
- How can humans and computerised tools cooperate in a coordinated and controlled way to support the software process?

Around *software process management and improvement*, several initiatives have been launched:

- *Industrial* initiatives to improve software processes have been adopted within software development organisations. Cooperations between academia and industry have also been established. A well-know initiative in this area is the Software Engineering Institute, created at Carnegie Mellon University in Pittsburgh in 1983. A more recent initiative is the European Software Engineering Institute, established in 1993 as a pan-European industrial action, partly supported by the Commission of the European Community.
- A new *research* area has grown up, supported by scientific events that facilitate the exchange and discussion of results, problems and early experiences [Der92] [Ost93] [Scha93].
- Outside the area of software production, similar efforts emerge for production processes within Information systems, Office automation, and various CAD/CIM systems. These may be carried out under names of *workflow support*, *groupware*, or *concurrent engineering*. So far, there has been little cross-fertilisation between these areas.

The recent intensive work on software processes has led to terminology and definitions of software processes which are rather confusing. As a consequence, it is difficult to compare different technologies and solutions, in order to understand their particularities and assess their specific merits and drawbacks. It therefore seems worthwhile to establish a common *reference framework* of (software) process concepts to be used within the software process community. The aim is to facilitate communication and assessment of emerging software process technologies, especially with respect to software process evolution (see below). Similar efforts have been done in the recent past (see [DNR91], [FH93], [CFFS92], and [Lon93]), but further work is still needed.

To properly take into account the different requirements and issues emerged in the above discussion, we propose to base such a framework on the following assumptions:

- A software process is composed of three parts: the actual *software production process*, the *software meta-process*, and the *software process support*.
- The software production process includes all the activities used to build, deliver, and maintain a software product.
- The software meta-process (or simply meta-process) includes all the activities related to the evolution of the whole software process, i.e., the software production process, the meta-process, and the process support. Its goal is to introduce into the software process all the innovations that can enhance the overall quality of the delivered products, of the software production process, and of the meta-process itself.
- The software process support is the computerised environment that can be used to support and, whenever possible, automate software process activities (both in the production process and the meta-process).

In our opinion, the meta-process is a crucial component in such a reference framework, and represents an effective support to software process improvement. An organisation (and the process through which it operates) must be seen as an organic entity, where the production process and the meta-process cooperate in a highly interacting way to evolve the whole process. Therefore, the meta-process must be built so that it can observe and manipulate the whole software process (including itself). This means that a software process has to be conceived as a *reflective system*.

Moreover, we can set up the following four abstraction layers in the production of an information system:

1. The meta-process to govern the entire process.
2. The production process to produce the software in the next point.
3. The delivered information system, being the software product (or artifacts) of the above production process.
4. The actual application (real-world process?) to be supported by the above software system.

Thus, what is the “real-world” process depends on the stand-point: it can be software production, or it can be banking, car assembly etc. being supported by the produced software.

Summing up, our work is based on the assumption that a software process is an organic entity whose goals are 1) to produce high-quality software products, and 2) to observe and improve its own structure according to its performances and the organisation’s overall strategy and objectives.

The paper elaborates and discusses this basic assumption and is structured as follows: Section 2 presents the basic concepts used throughout the paper, by synthesising current practice. Section 3 provides the fundamentals of the conceptual framework, by presenting a more detailed discussion of the meta-process, and of the concepts related to the evolution of a software process. Section 4 analyses the technological and methodological issues related to the development and support of the meta-process. Section 5 provides examples to put the presented concepts into practice. Finally, Section 6 provides some conclusions and points at possible applications of the proposed framework.

2 Basic Concepts

In this section we will present the basic concepts of the proposed framework. As a first step we informally introduce and motivate some basic definitions, and then we put them in a more general and systematic context.

A *software production process* can be coarsely characterised as a partially ordered network of interacting *activities*, aiming at producing a *software product*. Such activities are carried out by human *agents* (possibly organised in *teams*) supported by *tools*. They work upon software components and related documents, called *software items*. The set of software items constitutes what we call a *software product*.

The software production process needs to be *modeled* (i.e., explicitly represented) in order to be effectively repeatable, supportive (automatable),

and manageable. In particular, we want to be able to *control* a process and to *support* it. The former issue is related to the ability to ensure that the process evolves according to specific and desirable rules and procedures. The latter is related to the ability to automate production steps, and guide and solicit human activities.

Both the actual process and the corresponding *process model* need to be continuously evolved (modified, refined, or customised) in order to cope with different kinds of change requests. Thus, a software production process is part of an encompassing process, called *software process*, which also includes *meta-activities*, to support the evolution of the whole software process. We denote the set of such meta-activities with the term *meta-process*. This is described by a *meta-process model*, being a part of the software process model. The elements produced by the software process are denoted (*software process*) *artifacts*, and consist of software items (produced by the software production process) and software process models and tools in various forms (see Section 3.1).

To produce a model of a software process we need *software process modelling languages* offering suitable features to describe and manipulate the process artifacts¹. Parts of the same software process can potentially be adopted under different circumstances or by different organisations, so software process models are partly reusable. A *project* defines the context of a specific *instantiation* of a software process. A project context includes the profile of the product to be developed, scheduling, budgetary and organisational constraints, available resources, specific tools to be used, and any other relevant information that characterises the associated activities.

In the following, we will mostly consider processes in the *software* production domain, and the “software” prefix before process is often omitted.

Let us now provide a more systematic presentation of process-related definitions. The basic definitions are:

- **Process:** All the “real-world” *elements* involved in the development and maintenance of a product, i.e., *artifacts*, *production support* (tools), *activities*, *agents*, and *process support*.

It is composed of a *production process*, a *meta-process*, and a *process support* serving these two (with process modelling methods and language(s), process models, and process tools).

- **Artifact:** The (*sub-*)*products* and “raw material” of a process. An artifact produced by a process may later be used as raw material by the same or a different process to produce other artifacts. Artifacts are often persistent and versioned.

An aggregate of software artifacts to be delivered to a user is called a *software product*.

- **Activity:** A step of a process producing externally visible state changes to the software product. It incorporates and implements procedures, rules and policies, and aims at generating or modifying a given set of

¹Note that the software process model is an artifact and as such is one of the entities that are described using the process formalism. This aspect of self-reference is the fundamental reason for considering software processes as reflective systems.

artifacts. Activities may be organised into networks with both horizontal (chaining) and vertical (decomposition) dimensions. There may be both production activities and meta-activities. Activities are associated with *roles*, tools, and artifacts.

- **Tool:** Computer program supporting or automating a part of the work related to an activity. Tools are characterised by the operations they implement, their cost of use, and their availability. Tools may be further characterised according to the kind of interaction they support: *active tools* are able to start an interaction, for example to urge an individual to accomplish a specific action, while the interaction with *passive tools* is always initiated by humans.
- **Role:** A role describes a set of responsibilities, rights, and skills necessary to accomplish a specific activity in the software process.
- **Agent:** A human or specific person, performing the activities related to a role. An agent is characterised by the role properties and his/her availability.
- **Project:** Instantiation of a process to produce a specific product in a given organisation, with specific objectives and constraints.
- **Production support:** Set of methods, formalisms and related production tools to support the agents of a process to perform operations on artifacts.

For software, typical production tools are CASE tools, editors, compilers, reverse engineering tools, etc.²

The process modelling concepts are:

- **Production process:** Part of the process in charge of developing and maintaining the product to be delivered.
- **Meta-process:** Part of the process in charge of maintaining and evolving the whole process, i.e. the production process, its meta-process, and the process support.
- **Process model:** A description of a process expressed in a suitable *process modelling language*. A model is always an abstraction of the reality it represents, and is as such only a partial description. That is, there are always parts or aspects of the process, that are not captured in the model. The process model can e.g. consist of templates or types, and instances of these, to describe the various process elements.

There are two main submodels: the *meta-process model* and the *production process model*.

²Note that *businesses* other than software production can be modeled and supported using the framework presented in this paper, by substituting the software production support with other kinds of technologies. For example, information systems are processes in which humans and computer tools cooperate to achieve a specific business goal (e.g. to provide financial, banking or similar kinds of services). The *production support* used within these processes are EDP systems and similar computerised applications.

- **Process modelling language (PML):** Formal notation used to express process models, both for production processes and meta-processes. Since we want to model both the meta-process and the production process, the actual PML must offer *reflective features*. Non-reflective formalisms can model part of the process (typically the production process), but cannot be used to describe in a single, integrated entity both process and meta-process activities.
- **Process support:** A *process model* and the appropriate technology to define, modify, analyse, and execute it. The latter technology consists of process modelling methods and languages, *process tools* such as process modelling tools and process model interpreters (*process engines*), and the operating environment of such tools.

Summing up, the production process and the meta-process are entities of the *external* real-world, described through *internal* computer models to enable support and control. The computerised process models are manipulated by process tools to achieve the above control and support, and to evolve according to the evolution of the real-world.

All process elements can be considered versioned and composite objects. This raises the classical issues of Configuration Management (CM), namely change and consistency control, change propagation etc.. It is well-known that CM is a complex process that can benefit from an explicit process model to control activities related to updating, change propagation, auditing, rebuilding etc.. Conversely, the process elements themselves are (or should be) under CM control, with corresponding “meta-activities” – see next section and [CJM⁺92].

3 Software Process Evolution: the Meta-process

In the previous section, a process has been defined as the composition of a production process, its meta-process, and the corresponding process support. To fully understand the nature of “process evolution”, we have to understand how these components can evolve, both considering them in isolation, and taking into account possible mutual dependencies.

Firstly, it is obviously possible to change a production process without modifying the meta-process. This means that we change the way we conduct production activities, but we do not change the way we model and evolve them. Also, since models are partial, any change in the real-world process which does not effect the modeled parts, need not to be reflected by changes to the process support.

Secondly, we can improve or modify the meta-process, i.e., the way we model and evolve the process, without affecting the production process.

Thirdly, it is possible to change the process support, even if neither the production process nor the meta-process is changed. That is, both the process model and its associated tools may be extended (i.e. covering a larger part of the process), refined (i.e. giving more details), or made more powerful (e.g. giving better user guidance) – without needing to change the real-world process.

In summary, the production process, the meta-process, and their corresponding process models and tools may all evolve independently of one another.

3.1 Evolution of Process Support

To characterise the evolution of the process support, we need to better understand the nature and structure of a process model. Its own evolution will drive and thus evolve the entire process support [Con93].

A process model is composed of several **model variations**, holding refined or customised embodiments and representations of the same model. Variations can be classified into three different categories (or levels):

1. **Template** model variation:

This is the definition of the process model as expressed in the PML. Depending on the underlying paradigm of the language, a template may consist of type definitions, re-entrant code, or simply source code that may generate several run-time occurrences.

2. **Enactable** model variation:

This is a uniquely identifiable instance obtained from a template variation. Furthermore, it contains the necessary context-specific information to make it possible to execute³ it. Notice thus that at a given time, it is possible to have several enactable variations originated from the same template variation.

3. **Enacting** model variation:

When enactment actually starts, with a binding to a process model interpreter and with an explicit enactment state, an enacting variation is created from an enactable one.

The two relevant, primitive operations on variations are **create** and **modify**. The *enactment of a process model* involves six possible operations of creation and modification of the three model variations. Depending on the purpose of an enactment step, different operations are invoked as shown in Figure 1:

- When the purpose is to *apply* an existing model, new enactable and enacting variations are created.
- When the purpose is to *change* an existing model, this is accomplished either by creating or modifying a template, or by modifying enactable or enacting variations.

Other important, primitive operations to manipulate process model variations are:

- **Suspend** and **resume** enacting variations. The first operation suspends the execution of an enacting variation, so that its state and/or definition can be locally modified. Inversely, the variation can be resumed (restarted).

³In line with common practice, we use *enactment* as a synonym for process model execution or interpretation.

<i>Variation</i>	<i>Purpose:</i> Application of model	<i>Purpose:</i> Change of model
Template	none	create, modify
Enactable	create	modify
Enacting	create	modify

Figure 1: Operations on variations at model application and model change.

- Depending on the structure of the PML and the related support environment, variations can concurrently be modified by different process tools (process engines, editors, monitors). For this reason, suitable mechanisms have to be provided to support *atomic* operations on (sets of) variations, e.g. the classical **lock** and **unlock** to facilitate critical regions and related transactions.

Operations of model variations are implemented by the tools of the process support, e.g. the process language editor, the process model interpreter, the debugger for enacting variations etc.. Let us consider as examples the three process support systems the authors are involved in.

In **SLANG** [BFG93], there are multiple instances of a Process Engine tool, which create enactable and enacting variations (in an intertwined way) starting from the template variation of the process model. In **EPOS** [JC93], there is a Planner tool to create the enactable variation, and an Execution Manager tool to create and operate on (execute) enacting variations (the two tools work intertwined). In **Process WEAVER** [FER93], there is a process instantiation tool which copies a template, provides it with enactment parameters and links it with an interpreter to create the enactable variation. An enacting variation is created by a process starter tool and is then simply managed as a UNIX process. All the three process supports offer process model (“process schema”) editors to help creating and modifying template variations.

Evolution of the process support is more than simply creation or modification of template variations. Different scenarios can be envisaged, by considering different sequences of variation operations. We consider here three significant styles of evolution⁴:

- The first style is what we call *delayed change* of the process support. When we modify the template variation of a process model, we do not propagate the corresponding modification to the existing enactable and enacting variations. The modification is “seen” only when new enactable and enacting variations of the process model are created.
- The second style is what we call *busy change* of the process support. When we modify the template variation of a process model, we also propagate (the consequences of) this modification to the enactable and enacting variations generated from it.

⁴The reader should consider them as significant examples and not an exhaustive list.

- The third style is what we call *local customisation* of an enactable or enacting variation of a process model. In this case, we are interested in a change that does not affect the template variation of the process model, and which is just effective as long as the affected variation exists.

If the process is described by a single integrated model (a reflective system), these operations can be invoked from some other activity variation which is part of the *meta-process*. In general, therefore, the evolution of a process model can be carried out under the control of some enacting meta-variations. This fact is particularly significant when we shall consider the structure of the meta-process in the following.

An important comment concerns the granularity of the operations. For the sake of simplicity, we have implicitly assumed that a variation is associated with an entire process model. Actually, it is possible to extend or redefine the variation concept, e.g., to the activity level. That is, each activity can have a template variation and several enactable and enacting variations. This allows us to characterise real situations where a process model evolves through the invocation of operations applied to single activity variations in the process model.

Notice, also, that the borderlines between the different kinds of model variations may technically vanish when adopting specific technologies within the process formalism and the support tools. In particular, late/dynamic binding can facilitate incremental creation and evolution of template, enactable, and enacting variations, thus providing a high degree of flexibility and variability.

3.2 Evolution of Processes

The previous section was concerned with the evolution of process *support* (and specially models) in its various forms. Up to now very little has been said about the evolution of the process *itself*. That is, the focus has been to identify a new process that meets new requirements and to build a process model in sufficient detail to allow enactment support for such a new process.

Outside the scope of laboratories it is however unlikely that the new process, and the corresponding enactment of the new process model, can be installed without further preparation. Therefore, the meta-process is composed not only of the basic meta-activities to change the process support. It must include also all those meta-activities to analyse the existing process, in order to define the characteristics of the new target process, and to specify evolution from the existing process to this. The meta-process, in conclusion, must specify the policy that is used to manage the “transition” from the existing process and process support to the new ones. Moreover, it should allow deviations and temporary “bypasses” to adapt the meta-process itself to specific needs and situations.

There are numerous advantages of creating an explicit model of the meta-process. Firstly, such a model describes the strategy that has to be pursued to support a specific evolution step. If it is impossible to define the meta-process, it is also unlikely that the defined target process can replace the existing process.

Secondly, it may guide the preparation of the environment to accept the new target process. This preparation may be very far-reaching, and include technical aspects such as the introduction of new tools in the environment or the transformation of data to correctly represent new types and information structures. It can also include non-technical aspects such as user training, motivation, and the preparation of new user documentation.

Thirdly, it may be used to plan and to budget the introduction of the new process. Process improvement is ultimately measured in terms of cost. Benefits from changing an existing process must therefore be measurable either in the short term (e.g. reduced end-to-end time or resource costs), in the medium term (e.g. better sharing of resources or higher degree of reuse across projects), or in the long term (e.g. higher skills or better team motivation). However, benefits from improved processes must be put in relation to the cost of process change – i.e. the cost of the meta-process.

Finally, the meta-process model can help to measure progress in introducing a new target process: the state of the meta-process indicates how far a process improvement activity has come to reach its goal of installing a new process.

3.3 Structure of the Meta-process: Process Lifecycle

In the previous sections we have provided concepts to understand the nature of processes, both in the real-world and their corresponding models, and to characterise how they can be changed. As stated earlier, the meta-process is the process concerned with creating and evolving the process. This means that it must include all those activities that allow the agents to apply all changes to the existing process and its corresponding process support. We will therefore in this section describe the meta-process in terms of its principal constituent *meta-activities*.

Meta-activities can be built and composed in many different ways. It is not possible to identify one universal meta-process for all possible processes. However, it is possible to define some basic general meta-activities that constitute the skeleton of any meta-process:

1. **Technology provision**

This meta-activity is in charge of producing or acquiring the production support and the process support. Notice that if this involves the actual production of technology, this leads to a “process within the process”, and thus introduces a level of recursion in the process.

2. **Process requirements analysis**

This meta-activity takes into account the production process, the meta-process and the existing process support to provide (new) requirements to the design meta-activity. The resulting *requirements* specify (part of) the features and properties that the process has to offer.

3. **Process design**

When a new process is created, or if new requirements for an existing process are identified, this meta-activity provides the general and detailed architecture of the process. *Design specifications* might affect

different parts of the process, i.e. the production process, the meta-process, and/or the process support. An important input to this meta-activity is the description of the technology that has to be integrated and used within the process.

4. **Process implementation**

This meta-activity is in charge of implementing the design specification produced by the previous meta-activity. For this reason, this meta-activity can also be called *transient change meta-activity*. As described previously, this includes two conceptually distinct aspects:

- *Changing the process support* (Section 3.1)
If the design specification requires the creation/modification of the process support, its implementation is accomplished *by creating or modifying* model variations of the process model, possibly at fine granularity. In addition comes possible changes to the process tools.
- *Changing the real-world process* (Section 3.2)
If the design specification requires the modification/creation of the real-world process then this usually involves the definition of specific policies and strategies to manage the installation of the new process.

5. **Process assessment**

This meta-activity provides quantitative and qualitative information describing the performance of the whole process. Such information is used by the process requirements analysis meta-activity.

We want to stress once again that the proposed set of meta-activities does not impose any specific meta-process lifecycle (cf. waterfall model [Roy70]), but should give an indication of the fundamental steps that have to be taken in order to manage a process. Depending on the requirements and constraints of the development organization, it should be possible to support a meta-process following an evolutionary and incremental lifecycle, or also other non-traditional approaches.

As for the development of the production process support, there is an evident analogy between the meta-process of the process domain and traditional software production lifecycles. This suggests that we have to identify and adopt technologies and methods that might be quite similar, at least in principle, to what is used in software production. For instance, we might use executable specifications in both the analysis, design, and implementation meta-activities in order to support a meta-process based on a transformational lifecycle. A particular aspect of the software process domain, however, is its evolving and reflective nature, which imposes specific requirements on the software process support (see Section 4).

3.4 **Key Roles in the Software Process**

The following main roles can be identified in relation to the principal activities of the software process:

Process Owner: is responsible for the software process under consideration, provides informal process descriptions and evaluates process performance, thus producing additional feedback and requirements for process improvement. The process owner is the main source of information for the process analysis meta-activity.

Process Technology Provider: is responsible for providing the software process support, possibly using an existing process support environment. Meta-activity: technology provision.

Process Designer: is responsible for enriching the above process models, e.g. with template and enactable model variations. An important specialisation is the Meta-process Designer. Meta-activities: process requirements analysis and process design.

Process Manager: is responsible for the implementation of the software process and in particular the transformation of the template variation of the process model into its enactable and enacting variations. The process manager is also in charge of collecting data and information describing the performance of the software process. Meta-activities: process implementation and process assessment.

Process Agent: a human who operates within the software production process. A process agent uses the set of domain/application-specific rules, tools and technologies, all of which are controlled, assisted or enforced by the software process support. Meta-activities: mostly process implementation (enactment part).

4 Techniques and Methods for Process Evolution

As mentioned, a process consists of a production process and a meta-process, each consisting of “external” process elements (artifacts, agents, tools etc. – i.e., the real-world) and the “internal” process support. The latter again consists of a process model in different variations and of process model support tools.

A natural question is which aspects (what **process parameters**) of the above process elements and associated process support are inclined or allowed to change, and due to what reasons based on which inputs or feedbacks (why)? Further, how can the change be implemented and propagated, by which policy (how and when), and by which human roles (whom)? All this involves discussing the detailed requirements and architecture of the meta-process, its implementation and thus the specific procedures, rules and activities used to produce, enact and maintain the process.

In the following, we will first elaborate on the nature of the meta-process, and categorise some available techniques to manage process evolution. Then we sketch a framework for method support for such evolution and discuss some current method approaches. The focus will be evolution of models, not on process tools or external process elements.

4.1 Nature of the Meta-processes

The need for and the characteristics of process support and process changes will differ widely. Sometimes a fixed process is imposed by management; other times the process can change almost continuously by process agents, also acting as process managers, in a controlled manner. In the latter case, we want to treat meta-activities as flexible and integrated parts of the process. They need to be activated at appropriate times and points in the process lifecycle, because of feedbacks or incremental detailing of model definitions. They can often technically be modeled and implemented as normal production activities. Then what is so special about meta-activities?

From previous discussions, four observations naturally come out:

- **Meta-activities have deep semantics and may occur frequently:**

These activities may fundamentally affect process behavior, as their meta-artifacts have a large influence “range”. Frequent process changes also necessitate technologies like dynamic schemas and rapid prototyping (see next subsection).

- **Meta-activities must be carefully controlled:**

Regardless of the underlying machinery for change support, we need methods to structure and control the meta-process. We cannot allow undisciplined access or transformations of meta-models and meta-tools.

- **There is a potentially infinite recursion of meta-meta processes:**

If the meta-process is a normal process to be created, controlled, and maintained, what is the (meta-meta-)process to produce a meta-process, and so on? Like in any other reflective framework, we must let some of the meta-tools have embedded semantics to “root” this recursion. Note the analogy to bootstrapped systems, e.g. self-defining compilers with two “recursion” steps per new version of the compiler.

- **There are lessons to be learned from software engineering with respect to methods, techniques, and tools:**

What are the mechanisms, principles, methods, and policies we implement within the meta-process to produce (build and maintain) the process? Existing software production support (methods, formalisms, tools) should here be carefully assessed to verify their suitability within the given process and meta-process domain. Obviously, we need something similar to existing CASE tools for the domain of process support. This technology should support the development and maintenance of process models and their tools. Such integrated meta-tools would combine individual meta-tools and other technologies. They could coherently offer meta-activity support for model definition, analysis, creation, evolution, assessment etc..

4.2 Technologies to Support Process Model Evolution

To properly support process model evolution and transformation, we will probably need several kinds of mechanisms. The following five technologies seem the most important:

- **Flexible architectures** to support reconfigurable, modular, and distributed systems. This is needed in order to dynamically reconfigure the process support environment upon changes. In particular, we need *dynamic linking* and similar advanced features.
- **Reflection:** i.e., a PML should support late/dynamic binding and allow executable code to be manipulated as data⁵ (like in Lisp, Smalltalk, or Prolog). This is necessary to make it possible to dynamically reason upon, manipulate, and interpret a model (“code”), e.g. represented by “reified” classes and meta-classes [MN88] [FG93]. This is needed if we want to describe the process as a single, integrated model. Therefore, most process support environments – with highly diverse PMLs – rely on such techniques to achieve model flexibility during enactment.
- **Schema evolution and versioning:**
Technically we have to manage *evolving types/classes*:
 - versioned or modified classes, with all possible consequences for class instances⁶ (see below also);
 - subclasses or generic/parameterised classes;
 - instance-level delegation;
 - views;
 - ad-hoc, embedded binding mechanisms in the process support tools.

Schema evolution has been discussed extensively in the database literature [B⁺87]. It has a fundamental impact on persistent data, structured according to a given schema.

- **Impact analysis:** This is needed to assess the effect of changes before applying them. In particular, it is necessary to check that stated constraints are still satisfied after the change is applied, and to provide an estimate of change cost.
- **Restricting access rights:** Since process models are complex and structured, it is mandatory to have features supporting modularisation, interfaces, scoping, and association of *roles* to activities.

4.3 Methods and Principles to Guide Process Evolution

The previous subsection dealt with technical mechanisms to handle process support evolution. We shall now briefly study how we can define a method, i.e. meta-level rules, policies etc., to control and guide this evolution. We will assume that the *same* method which is used to define or provide the initial process support can *also* be used to evolve it later, as the definition of the process support is evolutionary. Namely, we need an *evolutionary production method* for processes. This method must cover the full lifecycle of a process: Technology Provision, Analysis, Design, Implementation, and Assessment. It is our opinion that no such total method

⁵It has viciously been said that process modelling has “rediscovered” interpretation.

⁶“Software items have longer life-time and are more stable than their models!” [Est92].

has reached sufficient maturity. What we can consider today are method fragments that may be applied to the individual meta-activities. Exploiting established methods from software engineering, we might come up with the following list:

1. **Technology provision:**

The various process support tools need to embody a significant number of the mechanisms described in Section 4.2. Due to their research-oriented nature, it should be expected that technology provision will be prototype-oriented with evolutionary introduction of new features.

2. **Process requirement analysis:**

Generally, we believe that conventional methods for analysing information systems (like SADT [Ros77], semantic modelling, object-oriented analysis) can be used for certain aspects. However, our current knowledge of especially the fine-grained process aspects is very low, and we will need to rely on combinations of knowledge acquisition techniques and formal techniques.

3. **Process design:**

We should expect that PMLs for design (and for implementation below) to embody known support for *modularity and reuse*. A future trend (once a significant number of process supported projects have been successfully completed) will be towards process design from reusable process fragments, and the role of the process designer will change character from being very technical (with good programming skills) towards more focus on quality methods and techniques.

4. **Process implementation:**

Process implementation is probably the most innovative area and the role of a process manager is quite new, with responsibilities somewhere between a quality manager and a project manager. Typical for the processes to manage is that they cross organisational boundaries, i.e. they involve several teams or sub-teams. Especially in the domain of process evolution will this activity entangle new methods and techniques, both technical and managerial.

5. **Process assessment:**

Here it seems that the emerging process capability methods such as the SEI CMM [PC+93] or the ESPRIT BOOTSTRAP approach [Boo93] will have their application. Moreover, the role of quantitative modeling methods based on product and process metrics is becoming crucial [AHM91].

We must regretfully conclude this section by saying that there is very little (new) method work on systematic support for process evolution, both in the production process and meta-process. A large repertoire of technologies is available, both for process and meta-process modelling and instrumentation, but a *complete* method base is missing. One significant example, however, of a method encompassing both the production process and the meta-process, is the *PRISM* model of change [Mad91]. This method suggests a general scheme to apply and track changes to different kinds of artifacts such as

software items and roles, but also process models and other meta-process related entities.

5 An Example: a Bank EDP Center

The EDP center of a bank is usually a large organisation, whose goal is to provide the bank departments with computer applications to support and conduct the bank's business, i.e. to offer efficient and valuable financial services to its customers.

The EDP center has a software process in which analysts and programmers (human agents) use software development tools and follow specific rules and procedures, in order to produce the above-mentioned software products. The management of the EDP center must coordinate and manage the different activities carried out within it, to ensure that the software products are delivered and maintained according to the EDP center's quality plan. Moreover, it has to maintain and evolve the process used to deliver such software products, i.e. it manages the meta-process. Such a meta-process is in charge of acquiring new software development tools (e.g. a new COBOL compiler or a reverse engineering tool), introducing new methods and approaches to software development (e.g. a prototype-based development), or updating the procedures used in the software process (e.g. modifying the Configuration Management operating procedures).

Both the production process and the meta-process can be described to better provide guidance and support to the EDP center's employees. Let us assume that the meta-process is not formally modeled and supported. Moreover, assume the process support includes only the model of a part of the software production process, namely CM activities. It is worthwhile to see how different kinds of changes then can be accomplished, and how they can be described using the conceptual framework presented in this paper:

- **Enriching the process with productivity measurements**

Let us suppose that we decide to introduce a measurement activity in the software process, to evaluate programmer productivity. This *requirement* is established by the managing director of the bank (process owner) who wants to have a clearer assessment of the productivity of the EDP center. In *designing* such a change to the process, the software process designer realises that it does not affect the current model of the software production process, since, at least in the first place, this does not deal with CM activities. He decides that the software production process has to be modified, by having programmers filling in an information form as soon as they release a source module. The *implementation* of such a design specification is accomplished by sending out a memorandum to all programmers, with the indications on how to accomplish the above-mentioned measurement activity, and by distributing a copy of the form.

- **Revising the process model by CM procedures for check-in**

Later on, the process designer may want to change the way a specific part of the CM activity is carried out. In particular, he/she want to add a step to the check-in procedure, so that an e-mail is automatically

sent to the programmer who is releasing a module, to invite him to complete the productivity form. He/she might also require that check-in operations that have not been terminated yet, are restarted using the new activity specification.

Using the terminology introduced in the previous sections, the above specification requires that, after we have modified the template variation of the check-in activity, we have to modify its enactable and enacting variations accordingly. To implement such a modification, therefore, we have to use a model editor to modify the template variation of the check-in activity, by adding the invocation of the e-mail tool. Moreover, we need to invoke some monitor/debugger tool to propagate the modification to ongoing check-in activities, i.e., existing enactable and enacting variations. The model editor and the monitor/debugger are both process meta-tools, used to create the software process support itself.

Notice that since the meta-process is not modeled and thus not managed by the software process support. It is up to the process manager/designer to properly invoke the editor and the monitor, in order to accomplish the required modification of the software process support.

- **Modelling the implementation meta-activity: busy vs. delayed propagation of model changes**

Finally, we may want to change the process support by introducing a model of the implementation meta-activity. We are therefore introducing a way to automatically support changes to the software process. As in the previous case, we have to invoke the editor to specify the new template variation describing the implementation meta-activity. Such a variation will contain a formal description of the steps we have been “manually” conducting so far, for example to modify the check-in activity model. In particular, it will contain the invocations of the model editor and the monitor, and will specify how to use them to accomplish the required evolution strategy (delayed change, busy change, ...). As soon as this editing activity is terminated, and the template variation of the implementation meta-activity is therefore available, it is possible to implement further changes to the process model by creating enactable and enacting variations of this newly created meta-activity. Eventually, we have “manually” introduced changes to the software process model to “automatically” support and guide future modifications.

It seems to us that this example demonstrates that the proposed framework can be fruitfully used to describe and analyse the evolution of a software process, i.e. the software production process, the meta-process, and the software process support. In particular, it is possible to describe the operations occurring in an integrated production process / meta-process system, according to a high-level reference model and using a set of coherent and consistent concepts.

6 Conclusions

The paper has presented a reference framework of concepts for (software) processes, with emphasis on their evolution. The main concepts are as follows: A software process consists of a production process, a meta-process, and the process support. The *production process* is the set of “external” production elements (real-world activities, artifacts, tools, agents, roles, and embedding project). The *meta-process* is the set of similar “external” meta-elements, that evolve and control the whole process. The *process support* is the “internal” process model and tools, that govern both the production process and the meta-process.

The meta-activities are divided into five phases in a process lifecycle: Technology provision, Process requirements analysis, Process design, Process implementation, and Process assessment. These phases can be active in a fine-grained and incremental way, e.g. based on feedbacks from later phases.

A process model describes the above-mentioned process elements, possibly at a fine granularity, and is dynamically and incrementally (re)elaborated and transformed in the process implementation meta-activity. It goes from a *template model variation*, via an *enactable model variation*, to an *enacting model variation*.

The framework is applicable for technology providers and technology users. It should not act as a mere check-list, e.g. does system X offer formal “meta-process tools” or “enacting template models”. Rather, it should provide a systematic way to understand, assess, and compare process support systems. We can compare it with the Object-Oriented Database System manifesto [ABD⁺89].

Section 1 in the paper points at some related work. Much underlying technology can be taken and adapted from general software engineering, including dynamic instrumentation and binding facilities.

However, we still miss a *reference method* for process evolution that covers non-trivial applications, and that is coupled to tools and roles for the process actors of the meta-process. We also need better understanding of the process implementation issue, especially with respect to managing process evolution.

The framework will be evaluated in the ESPRIT Basic Research Action #7082, PROMOTER, where it will be applied to different scenarios and process management systems.

Acknowledgements

Thanks go to the participants of the 2nd European Software Process Technology Workshop and to our partners in PROMOTER. In particular, we want to acknowledge the contribution of Bob Snowdon at ICL, who has provided us with many valuable ideas and comments. Individually, the authors also want to express their thanks to colleagues at Politecnico di Milano, CEFRIEL, NTH, Cap Gemini Innovation and in the Eureka Software Factory project.

References

- [AHM91] Tarek K. Abdel-Hamid and Stuart E. Madnick. *Software Project Dynamics*. Prentice-Hall, 1991, Englewood Cliffs, New Jersey.
- [ABD⁺89] Malcolm Atkinson, Francois Bancilhon, David DeWitt, Klaus Dittrich, David Maier, and Stanley Zdonik. The Object-Oriented Database System Manifesto. In *Proceedings from DOOD'89*, Kyoto, Japan, Dec. 1989, 40–57.
- [BCN90] Hugh R. Beyer, Kathy Chapman, and Chris Nolan. The ATIS reference model. Technical Report ZK02-3N30, Digital Equipment Corp., 110 Spirit Brook Rd., Nashua, NH 03062, April 1990.
- [BFG93] Sergio Bandinelli and Alfonso Fuggetta and Carlo Ghezzi. Software Process Model Evolution in the SPADE Environment. *IEEE TSE*, December 1993, 37 p.
- [B⁺87] Jay Banerjee et al. Semantics and implementation of schema evolution in object-oriented databases. In *Proceedings of ACM SIGMOD '87*, pages 311–322, May 1987.
- [BL79] L.A. Belady and M. M. Lehman. Characteristics of large systems. In Peter Wegner, editor, *Research directions in Software Technology*. MIT Press, 1979.
- [Boo93] The Bootstrap Project Team. Bootstrap: Europe's Assessment Method. *IEEE Software*, pages 93–95, May 1993.
- [CJM⁺92] Reidar Conradi, M. Letizia Jaccheri, Cristina Mazzi, Amund Aarsten, and Ngoc Minh Nguyen. Design, use, and implementation of SPELL, a language for software process modeling and evolution. In [Der92], pages 167–177.
- [CFFS92] Reidar Conradi, Christer Fernström, Alfonso Fuggetta, and Bob Snowdon. Towards a Reference Framework for Fundamental (Software) Process Concepts. In [Der92], pages 3–17.
- [Con93] Reidar Conradi. Resumé from Session on Process Change at ISPW'8. In [Scha93], pages 18–21.
- [Der92] Jean-Claude Derniame, editor. *Proceedings Second European Workshop on Software Process Technology (EWSPT'92)*, Trondheim (Norway), 7–8 Sept. 1992. LNCS 635, Springer Verlag, 253 p.
- [DNR91] Mark Dowson, Brian Nejme, and William Riddle. Fundamental Software Process Concepts. In [FCA91], pages 15–37.
- [Est92] Jacky Estublier. In discussing dynamic typing in the Adele versioned software engineering database. At EWSPT'92, Trondheim, Sept. 1992.
- [FER93] Christer Fernström. Process WEAVER: Adding Process Support to UNIX. In [Ost93], pages 12–26.

- [FH93] Peter H. Feiler and Watts S. Humphrey. Software Process Development and Enactment: Concepts and Definitions. In [Ost93], pages 28–40.
- [FCA91] Alfonso Fuggetta, Reidar Conradi and Vincenzo Ambriola. *Proceedings of the First European Workshop on Software Process Modeling (EWSP'91)*, Milano, 1991. AICA (Italian National Computer Science Society), 272 p.
- [FG93] Alfonso Fuggetta, Carlo Ghezzi. Process Formalisms Need to be Fully Reflective. In [Scha93], pages 78-80.
- [JC93] M. Letizia Jaccheri, Reidar Conradi. Techniques for Process Model Evolution in EPOS. Accepted for special issue of IEEE TSE, December 1993, 18 p.
- [Lon93] Jacques Lonchamp. A structured conceptual and terminological framework for software process engineering. In [Ost93], pages 41–53.
- [Mad91] Nazim H. Madhavji. The process cycle. *Software Engineering Journal*, 6(5):234–242, September 1991.
- [MN88] P. Maes and D. Nardi, editors. *Meta-Level Architectures and Reflection*. North Holland, 1988.
- [Ost93] Leon Osterweil, editor. *Proceedings of the Second International Conference on Software Process (ICSP-2)*, Berlin (Germany), February 1993. IEEE-CS Press.
- [PC+93] M. Paulk, B. Curtis, M.B. Chrissis, and C.V. Weber. Capability Maturity Model for Software, Version 1.1. CMU/SEI Technical Report, No. 93-TR-24, February 1993.
- [Ros77] D. T. Ross and K. E. Schuman. Structured Analysis for Requirements Definition. *IEEE Trans. on Software Engineering*, SE-3(1), pages 16–34. January 1977.
- [Roy70] W. W. Royce. Managing the Development of Large Software Systems: Concept and Techniques. In *Proceedings of WesCon*, August 1970.
- [Scha93] Wilhelm Schäfer, editor. *Proceedings of the Eight International Software Process Workshop*, Dagstuhl (Germany), 2–5 March 1993. IEEE-CS Press.