Published in: Proceedings of the 8th International Conference on Software Engineering and Knowledge Engineering, June 10-12 1996, Lake Tahoe, Nevada, USA

A Synthesis of Two Process Support Approaches*

Martin Verlage, Barbara Dellen, Frank Maurer, Jürgen Münch

Fachbereich Informatik, Universität Kaiserslautern, Postfach 3049, 67653 Kaiserslautern, Germany
{verlage, dellen, maurer, muench}@informatik.uni-kl.de

Abstract. *Process support approaches use process models to allow for communication, reasoning, guidance, improvement, and automation. Two approaches are compared and combined to build a more powerful one. In this paper we describe the synthesis of the CoMo-Kit and MVP-E approaches. CoMo-Kit is based on AI/KE technology. It was developed for supporting complex design processes and is not specialized to software development. MVP-E is an environment for modeling, analyzing, and guiding which additionally provides services to establish and run measurement programmes in software organizations. Differences between both approaches are identified which are then used as alternatives when deriving requirements for process support environments. The discussion helps understand what is intrinsic to processes in general and what is special to software development processes in particular. Synthesizing both process support systems, which were developed completely independently, allows to alternate planning and enaction of software development processes.*

1 Introduction

Processes are present whenever information is created, transformed, or communicated. Although we distinguish between different types of processes (e.g., business processes, decision processes, and software development processes) these types share common properties. Understanding commonalities and differences between process types is a key factor for better process support. Process support includes improved communication, detailed reasoning about process features, guiding people when performing processes, improving both processes itself and their results, and automating process steps in order to gain deterministic process behavior [9, 26]. These purposes require explicit process representations (i.e., *models*).

The variety of existing process support systems, for example process-sensitive software engineering environments or workflow management systems, corresponds to the variety of process types. In this paper we focus on software development processes and software processes. A *software development process* is a (sub-)set of technical activities to build a *software product*. A *software process* includes all software development processes and all organizational processes needed to drive the project (e.g., managerial processes). The addition 'software' may be omitted in this paper for the sake of brevity. Interpretation of development process models, performed by a *process engine*, is called *enactment* to emphasize necessary user participation.

Process engines, as part of process support environments, should have mechanisms for replanning a project under enaction, which addresses a main problem of *process evolution* [19]. Without these features such an environment is not applicable in real-world projects. Unfortunately, only limited solutions do exist ([19], p. 1126). Providing such a powerful environment is one goal of our work. Two already existing approaches from knowledge engineering and software engineering domains are synthesized. The Conceptual Model Construction Kit (CoMo-Kit) was developed for supporting complex design processes (e.g., city planning). The Multi-View Process Environment (MVP-E) supports modeling, analyzing, and guiding. Additionally it provides services to establish and run measurement programmes in software organizations. Both research prototypes were developed completely independently. By studying them, relating their underlying assumptions, and synthesizing them we gain a better understanding about the principles of process support environments. The synthesized approach should allow to prescribe development processes without hindering creativity. Necessary restrictions and rules delineate human processes performance. It is intrinsic to software development processes that at any time in the project the next steps can be described precisely whereas the latter steps do not have a sharp shape. Process support environments should provide mechanisms to adjust the project plan from time to time. Moreover, capturing dependencies during product evolution allows goal-directed backtracking in order to bring the project back to the right track. Although this vision is still to be achieved, first results of synthesizing CoMo-Kit and MVP-E already exist. They concentrate primarily on technical issues rather than providing support for project managers.

*This work is supported by the Deutsche Forschungsgemeinschaft as part of the Sonderforschungsbereich 501 "Development of Large Systems with generic Methods".

Published in: Proceedings of the 8th International Conference on Software Engineering and Knowledge Engineering,
June 10-12 1996, Lake Tahoe, Nevada, USA

A first step of synthesis carefully compared both approaches. The findings are promising and have been the basis for the next steps of synthesis, namely the integration of both systems. Although the integration is not completed, the comparison of process support approaches provides interesting observations about software development processes. We identified commonalities and differences of both approaches. This is done with respect to the underlying assumptions of each approach to help understand what is intrinsic to processes in general and what is special to software development processes in particular. The results match related findings (i.e., software process frameworks). Moreover, uncovering the underlying assumptions of CoMo-Kit and MVP-E explains their differences. This leads to important principles for modern process support. The principles will guide the development of a process-sensitive software engineering environment which allows for alternating planning and enactment activities.

The paper is organized as follows: The concepts implemented in MVP-E and CoMo-Kit are surveyed and compared to related work in Sections 2 and 3. This survey is needed to understand the discussion of commonalities and differences of the approaches in Section 4. Section 5 identifies requirements for process support environments by addressing the differences. In Section 6 we compare our findings to related research work. Finally, Section 7 summarizes the paper.

2 MVP-E

The MVP project began at the University of Maryland and continues at Universität Kaiserslautern, Germany. Its goal is to provide MVP-E, an environment as an instance of the ideas developed in the TAME project [5]. Special attention is paid to measurement activities which are essential when the environment should be used by all roles of the project and the organization [18]. MVP-E supports modeling, planning, simulation, enactment, measurement, recording, and packaging of software engineering processes. The main idea is to split descriptions of software development processes into views. Every view is generally defined as a projection of a process model that focuses on selected features of the process [29].

The language MVP-L is used to describe development processes. It distinguishes between processes, products, resources, and their attributes which correspond to measurable qualities of the objects; processes, products, and resources are instantiated with respect to types and related by a project plan [7]. A second notation is used to represent GQM trees which are specifications of measurement goals from a specific viewpoint [4]. The discussion of GQM is beyond the scope of this paper. MVP-L has been evaluated in several industrial settings (e.g., [15]). The case studies' feedback became input for evolution of MVP-L. Table 1 summarizes the main concepts of MVP-L.

Concept	Explanation
Process	Activities which create, modify, or use a product.
Product	Software artifact describing the system to be delivered.
Resource	Human agent or tool.
Attribute	Measurable characteristic of a process, product, or resource.
Criteria (entry, invariant, exit)	Expression which must be true when starting, enacting, or terminating a process respectively.
Refinement	Breaking down the structure of a process, product, or resource into less complex parts.
Instantiation	Creating an instance of a process type, product type, or resource type and providing actual parameters.
Product Flow	Relationship between processes and products. It is distinguished between reading, writing, and modifying access.
Resource Allocation	Assigning personnel to processes in order to perform the processes.
Process Model, Product Model, Resource Model	Type descriptions of processes, products, and resources respectively. The name *Model* might be misleading in this context, because their instances are models of real-world processes, too.

Table 1: Concepts of MVP-L

MVP-E's process engine MVP-S uses a project plan and process models (types) to build its own representation of a real-world project [18]. The process engine is used to manage project data and to guide developers in their work. Processes can be enacted if they are *enabled* (i.e., their entry criteria are true). The agents, represented as resources assigned to processes, are responsible for achieving the goals specified in the exit criteria without invalidating the process invariants. Throughout enactment measurement data are taken which are used by all project roles (e.g., testers, managers) to reason about the project and to trigger actions. The following assumptions (labeled as **AM*i***) were made during the evolution of the MVP-E approach:

AM1: The concepts implemented in MVP-L are suited to express a single view on a particular software process.

AM2: Attributes of processes, products, and resources are sufficient for all measurement purposes.

Published in: Proceedings of the 8th International Conference on Software Engineering and Knowledge Engineering, June 10-12 1996, Lake Tahoe, Nevada, USA

AM3: Developers are guided by enacting understandable process models.
AM4: All project roles are supported by using the same representation of a project. The roles get own presentation of the project (i.e., views), tailored to their specific needs.
AM5: Products are represented incomplete. Direct access to the product is not supported. Only product models are accessed.
AM6: The steps of modeling, planning and enaction are sequential. Project plans remain unchanged over the project lifetime.
AM7: Planning provides parameters which adapt process models to different contexts.
AM8: Plan deviations are not considered. No support for modifying the project's state is provided.

Several other approaches for supporting software development processes have been developed [1, 9, 26]. In general, we can distinguish between languages for modeling fine-grain processes (i.e., used to integrate tools, hence being similar to programming languages) and coarse-grain processes (i.e., used to guide software developers and to coordinate their tasks, hence being similar to specification languages). Currently, the community focuses on the latter kind of processes. An example is the evolution of the Marvel Strategy Language, which was first designed for use in a single-user system and now supports distributed teams [6]. MVP-L belongs to the second category of languages. Another classification of software process languages can be made with respect to their level of abstraction. Granularity ranges from abstract levels with rich semantics of the concepts (e.g., MVP-L) down to detailed levels which provide powerful mechanisms to build one's own process building blocks (e.g., APPL/A [27]).

Not all languages cover an equal set of aspects of software development processes. Mostly, the languages provide a solution for a particular problem and support only a limited set of roles [25]. For example, the language SLANG focuses on capturing the dynamic aspects of a process [3]. MVP-L addresses process interfaces, rule-based specification of development processes and measurement aspects. Quality considerations have become more and more important within software development. However measurement aspects have not been taken into account in most of the process support environments. Approaches to support measurement activities within software development already exist [17]. MVP-E is an environment which includes measurement support.

3 CoMo-Kit

The CoMo-Kit project at Universität Kaiserslautern aims at support for planning, enacting, and controlling complex workflows in design processes from a knowledge engineering perspective [20]. Properties of such workflows are:

- They are too complex to be planned in detail before enactment starts. Results of activities are needed to plan later steps. Planning and enacting must alternate during the whole project.
- Decisions are made during enactment which rely on assumptions that can be found invalid afterwards. When old decisions change, users must be supported in reacting to the new situation (i.e., backtracking must be supported).

The CoMo-Kit system consists of a tool which allows to model complex workflows and a workflow management system, the CoMo-Kit Scheduler, which enacts and controls the modeled workflows. To describe complex design processes, the CoMo-Kit methodology uses four basic concepts: tasks, methods, concepts, and agents. Table 2 contains an overview and short descriptions of the terms. Using these concepts generic software process models and concrete project plans can be described. The next question to be answered is how to enact these plans. The enactment of design processes is supported by a flexible workflow management system, the CoMo-Kit Scheduler.

In Table 3 concepts of the Scheduler are explained. The main features of the Scheduler are:[1]

- The Scheduler allows alternate planning and enactment of development processes.
- Based on the information flow between tasks, causal dependencies are acquired during enactment. The underlying assumption is that the inputs of a task influence the outputs. For every task a set of logical implications is created; the implications relate the assignment of values (i.e., products) to the input parameters of a task with the assignment of values to the output parameters. Whenever the assignment of an input parameter becomes invalid, the assignments of the output parameters become invalid too. The causal dependencies improve the traceability of design decisions and support the users in reacting to changes.
- Additionally, the scheduler manages dependencies extractable from the task decomposition. Whenever a task becomes invalid because of a replanning of the project, the Scheduler notifies team members working on subtasks that they can stop working on them.
- To handle dependencies efficiently, reason maintenance techniques are used [23,24] and extended. The Scheduler

1.The techniques which are used to get these features are beyond the scope of this paper. For a detailed description see [21]. For a description from a software engineering point of view see [10].

Published in: Proceedings of the 8th International Conference on Software Engineering and Knowledge Engineering, June 10-12 1996, Lake Tahoe, Nevada, USA

uses the acquired justification structures to support dependency-directed backtracking.

Concept	Explanation
Task	A description of the goal which should be reached by an activity.
Input Variables	Information which is needed to work on a task.
Output Variables	Information which is the outcome of working on a task.
Method	A description of how a task's goal can be reached. For every task a set of alternative methods can be described.
Atomic Method	Atomic methods assign values to the output variables of the related task.
Complex Method	Complex methods decompose a task into subtasks.
Information Flow	A complex method is described by an information flow graph which consists of (sub-) tasks and variables. The information graph shows the input/output relations of tasks.
Concept Class	A description of the structure of the information (product structure) which is produced during enactment.
Slot	Stores part of product information.
Concept Instance	Concrete information, for instance a product (such as requirements document) which is outcome of using a method to solve a task.
Agent	An actor who works on tasks. Agents apply methods to solve tasks.

Table 2: The CoMo-Kit modeling framework

The following assumptions (labeled as **AC*i***) were made during the evolution of the CoMo-Kit approach:

AC1: Modeling, planning, and enacting design processes cannot be separated. They should alternate.

AC2: Project plans have to be modified and refined during project lifetime.

AC3: The Scheduler manages the state of the project. Therefore, it should have access to all products and manage them.

AC4: Users must be supported in keeping track about where the result of their work is used and what information they need in producing their results.

AC5: Changing old decisions is inherently needed in complex projects.

AC6: Causal dependencies are the basis for an active notification mechanism which informs users about relevant changes in the project state.

AC7: Only processes performed by individuals are modeled in order to keep track of dependencies.

AC8: The same formalism can be used to describe generic process models and concrete project plans.

Concept	Explanation
Decision	To solve a task (i.e., to reach the goal) an agent has to decide which method should be applied.
Assignment	Applying atomic methods to tasks results in the assignment of values to the output variables of tasks.
Task Decomposition	Applying complex methods to tasks results in a set of subtasks which are included into an agenda. This agenda stores a list of all tasks which must be solved to finish the project.
Dependency	From the information flow a set of causal dependencies is derived. We assume that there is a causal dependency from the inputs of a task to its outputs. The Scheduler manages dependencies.
Decision Retraction	During project enactment decisions can be found erroneous which results in an inconsistent project state. Then, at least one decision must be retracted and replaced by an alternative.

Table 3: Concepts of the CoMo-Kit Scheduler

Several other approaches for supporting workflows have been developed. In [12] an overview on current workflow management techniques is given. The authors show that current technology assumes that the process model is defined before process enactment. The CoMo-Kit approach was developed for application domains where planning and enactment alternate, for example design processes (e.g., city planning). Some workflow management systems support object-oriented data structures. For instance in [22] an innovative approach is described which integrates Petri nets with semantic data models. CoMo-Kit also supports object-oriented data structures. Further, CoMo-Kit supports reacting to decision changes.

4 Commonalities and Differences

Synthesis of CoMo-Kit and MVP-E requires the recognition of similar concepts. It is interesting that on the one hand the concepts of CoMo-Kit are a subset of those present in MVP-L. This is a hint that both approaches contain concepts that are intrinsic to processes in general. On the other hand, the functionality of the CoMo-Kit process

Published in: Proceedings of the 8th International Conference on Software Engineering and Knowledge Engineering,
June 10-12 1996, Lake Tahoe, Nevada, USA

engine exceeds MVP-S by alternating modeling, planning, and enaction and supporting change processes.

As shown in Figure 1, in an integrated process-sensitive software engineering environment, CoMo-Kit's task will be to trace the project, especially to document the decisions for rejecting an alternative of the plan, and to support short-term planning. Moreover, incorrect decisions may be retracted at a later point in time and backtracking is then performed to regain a valid project state and choose another alternative of the project plan. MVP-S, already designed for software process issues, is used for long-term planning and for measurement of different kinds of entities.

4.1 Commonalities

It is not surprising that both CoMo-Kit and MVP-E provide similar process and product concepts (see also Table A.1 of the appendix). Products and processes may be refined into subprocesses and subproducts respectively. In CoMo-Kit a separation is made between the goal of a process and the way how this goal can be achieved. In MVP-L both concepts are combined. Processes are performed by human resources. Refinement of processes and products are specified very similar in both approaches. The objects are arranged in a refinement hierarchy.

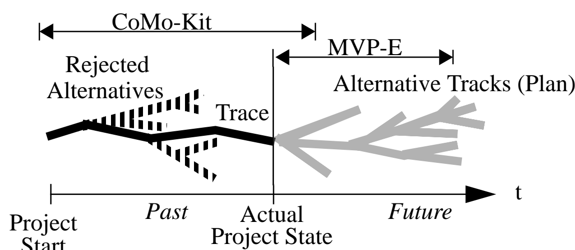

Fig. 1: Domains of CoMo-Kit's and MVP-E's Process Engines

Of course, implementations of concepts developed independently are not likely to match completely. Hence the concepts discussed in this subsection show some implementational differences (e.g., maximum length of identifiers, scoping rules, or use of instantiation parameters). But they are classified as minor and can easily be removed when integrating both approaches.

4.2 Differences

The overlap of concepts is a prerequisite for synthesis. Differences between MVP-E and CoMo-Kit (labeled as **D*i***) are caused by the different assumptions made for both approaches. They are listed for each D*i* at the end of each paragraph.

D1: *Instantiation.* MVP-L provides more powerful concepts for instantiating objects than CoMo-Kit. For example, initial attribute values can be computed using an expression to combine formal parameters of the object. In CoMo-Kit the user is expected mainly to provide values at runtime. In fact, CoMo-Kit's mechanisms are not so complex because the system allows dynamic instantiation. (AM7, AM8, AC1, AC2)

D2: *Agents.* CoMo-Kit assumes that exactly one agent performs a process. Several agents, one of which is the *process owner*, are allowed to perform a process in MVP-S, so that teams can be assigned to a process. (AM4, AC7)

D3: *Tools.* CoMo-Kit does not allow supporting tools to be modeled, which is possible using MVP-L. (AM1, AM3)

D4: *Type Hierarchies* are not expressible using MVP-L. No inheritance mechanisms are provided. Thus, specialization and generalization cannot be specified in MVP-L in contrast to CoMo-Kit. (AM1, AM7, AC8)

D5: *Attributes* of processes and resources are present in MVP-L models but absent in CoMo-Kit process representations. Only products may have additional characterizing elements. (AM2)

Because processes are the major concept in both systems and both approaches aim at different goals, it was not surprising to determine that the essential differences belong to processes. In addition to those existing for products and resources, there were the following differences:

D6: *Product Flow* between processes is specified in both approaches. Writing and reading access to products is expressible in CoMo-Kit and MVP-L. In addition, MVP-L allows modification of products by processes other than the producing one. Thus, multiple processes can evolve a product. (AM1, AC4)

D7: *Control Flow.* MVP-L provides constructs for entry and exit criteria (i.e., pre- and postconditions) to specify starting and terminating a process instance. CoMo-Kit realizes a pure data driven strategy which means a process may start when the products it accesses are present. A method is applied when an agent accepts the task. (AM3)

D8: *Process Iteration.* The differences D6 and D7 do not have only relevance for the modeling of processes. In MVP-S multiple enactions of a process instance are possible, whereas in CoMo-Kit a process never is enacted more than once. Thus MVP-S allows loops of process enactions, and CoMo-Kit does not. A loop is expressed as shown in Figure 2. (AC4, AC6)

The process engines of CoMo-Kit and MVP-E (i.e., the Scheduler and MVP-S) demonstrate necessary features of process support environments, but address different aspects (see also Figure 1). Their main differences are:

Published in: Proceedings of the 8th International Conference on Software Engineering and Knowledge Engineering, June 10-12 1996, Lake Tahoe, Nevada, USA

D9: *Planning and Enactment.* MVP-S requires a complete project plan at the beginning of a project. Only limited constructs exist to specify generic process elements. The Scheduler allows alternation among modeling, planning, and enaction of processes. (AM6, AM8, AC1, AC2, AC6, AC5)

D10: *Role Support.* MVP-S offers interfaces for organizational processes [18]. Those processes offer support to development processes (e.g., configuration management), or initiate remodeling and replanning. The Scheduler supports primarily software developers who perform process models and offers predefined views for project planners. (AM4)

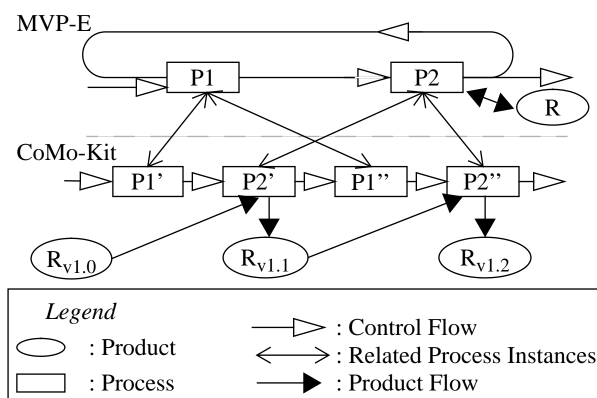

Fig. 2: Two Views of Process Instances

D11: *Product Management.* CoMo-Kit manages the products itself, i.e., the files which store product information are handled. In contrast, MVP-E operates on representations of the products. Even products not present in the computer (i.e., paper documentation) can be used for reasoning about project states. This requires a disciplined interaction between the user and the system to avoid a mismatch between the real world and the model world (i.e., the representation). (AM5, AC3, AC6)

D12: *Backtracking.* Plan deviations of the project cannot be managed by MVP-E. A major feature of CoMo-Kit is its ability to perform backtracking from each state to former project states. Another alternative is selected with or without changing the models and the project proceeds in a new direction. (AM8, AC2, AC5)

The above identified differences mark potential incompatibilities of CoMo-Kit and MVP-E. Some features are pure extensions (i.e., D1, D3, D4, D5, D7, D10, D11, and D12) and others are conflicting (i.e., D2, D6, D8, and D9). Removing the differences is described in the next section.

5 A Unified View

Section 2 and Section 3 presented assumptions which guided the evolution of two process support approaches. The assumptions can be regarded as requirements for a process-sensitive software engineering environment. Section 4 explained which of the assumptions conflict. The following list of requirements for process support environments reformulates the assumptions made in both approaches. Incompatibilities are removed, so that the requirements represent a unified view. The set of requirements is not to be understood as an exhaustive list of features a process-sensitive software engineering environment should have. The requirements were defined from the perspectives of both systems CoMo-Kit and MVP-E, nevertheless they are mandatory for each environment to support real development processes. But aspects are still missing (e.g., resource scheduling, time planning, budgeting). For each requirement (labeled as **R*i***) the related assumption(s) (AM*i* or AC*i*) are given.

R1: *The main concepts of software development processes should be provided.* (AM1, AC7, AC8) Algorithms for planning, analyses, or enactment need to know the semantics of basic entities (e.g., processes, products, product flow). It is important to tailor the concepts' representation to human and machine understanding. MVP-L addresses a sufficient set of concepts (see also the appendix) [7]. D4 (*Type Hierarchies*) is addressed by providing mechanisms for building type hierarchies (e.g., single inheritance) which enables a more comfortable handling of model variants. D6 (*Product Flow*) is addressed by version management. From CoMo-Kit's point of view a modification is just the creation of a new object (in Figure 2 the modification of R corresponds to the creation of $R_{v1.1}$ and $R_{v1.2}$ from $R_{v1.0}$ and $R_{v1.1}$ when enacting P2). To handle different versions of process models requires sophisticated configuration management. D7 (*Control Flow*) is addressed by allowing criteria to specify conditions for starting and terminating processes. This requires modification of CoMo-Kit's process engine.

R2: *Organizational processes should be supported.* (AM4) Specific interfaces to organizational processes are required. Further development in the area of multiple views on a project representation is needed in order to gain role-specific representations [29]. D2 (*Agents*) is addressed by allowing teams to perform a project. This is emulated in CoMo-Kit by representing teams as special users. Mechanisms for managing team structures are needed. D10 (*Role Support*) is addressed by using MVP-E's support for organizational activities.

R3: *Both short-term planning (detailed level) and long-term-planning (abstract level) are supported.* (AM7,

Published in: Proceedings of the 8th International Conference on Software Engineering and Knowledge Engineering, June 10-12 1996, Lake Tahoe, Nevada, USA

AC1) It is desirable to plan activities early in a project, but it is not practical in every case. Mechanisms are needed on the one hand for planning abstract and general dependencies and on the other hand for planning concrete and state-dependent ones. D1 (*Instantiation*) is addressed by using the features of MVP-L for instantiation, which include those of CoMo-Kit.

R4: *Allow for alternating modeling, planning and enaction.* (AM6, AC1, AC2) Information for planning is incomplete when launching the project. Required information may be produced during the project. Later planning steps refine the models or the original plan. D9 (*Planning and Enactment*) is addressed by using CoMo-Kit's part of the process engine which supports modeling, planning, and enaction, which leads to a new process state in MVP-S.

R5: *Document decisions.* (AC4, AC6) Many decisions about products and processes need to be documented [27]. The decisions explain how products evolve and processes are performed. The decisions are used to recognize deviations and to signal inconsistencies to the system. Dependencies cannot only be used for backtracking but also for reasoning about products [10]. D8 (*Process Iterations*) is addressed by tracing the information on the more concrete level (e.g., P1', P1'' in Figure 2) instead of the abstract one (i.e., a trace is acyclic). Nevertheless, an abstract view may also be offered to the user.

R6: *Reactions on changing decisions.* (AC5) When an invalid state is reached after retracting a decision, backtracking should be performed in order to reach a branch where an alternative can be chosen. Though D12 (*Backtracking*) is addressed by using the mechanisms of CoMo-Kit.

R7: *Manage the products whenever possible.* (AM5, AC3) Inconsistencies between the real-world project and the representation managed by the environment must be avoided. MVP-E works purely on representations, but CoMo-Kit manages (part of) the products. D11 (*Product Management*) is addressed by using CoMo-Kit's realization because it is a pure extension of MVP-S's representation approach (a product is always its own representation).

R8: *Support measurement activities.* (AM2) Quantitative data from processes and products are needed in order to develop software systematically [26]. Measurement, evaluation, and storage of data is needed. In MVP-S several mechanisms for management of measurement data are implemented [18]. D5 (*Attributes*) is addressed by using the features of MVP-L. At the moment it is not clear how to handle measurement data in the case of backtracking. For example, effort data should be kept but statements about products (e.g., subjective classification of complexity) might become invalid.

R9: *Software developers need to be guided.* (AM3, AC4) Process models explain what activities to perform next and what their goals are. Both CoMo-Kit and MVP-E provide graphical interfaces to visualize project data [20,18].

R10: *Software developers' tasks need to be coordinated.* (AM4, AC6) Process models are used to relate tasks of software developers. In the case a result becomes invalid, coordination means notifying others to interrupt their work. Both CoMo-Kit and MVP-E allow coordination on basis of explicit process models.

R11: *Execution of process fragments.* Process programs are used to automate process steps. It must be ensured that the dependencies established during a tool invocation are captured and documented in the environment's repository. This requirement was not a focal point in both approaches. CoMo-Kit provides a simple mechanism, scripts, to execute process fragments. MVP-S provides an interface to measurement tools to get data about products or to prompt for user data (e.g., effort spent for a particular process). D3 (*Tools*) is addressed by allowing tools to be modeled and supporting access to them when a developer performs the process.

The first ten requirements describe features of either CoMo-Kit or MVP-E. Automatic execution of process programs is a necessary, but straightforward to realize next step. Building a system which fulfills all these requirements is the challenge of integrating CoMo-Kit and MVP-E. Nevertheless, there are still open problems which are not tackled by the synthesis of both approaches:

P1: *State Manipulation.* When a project deviates from the plan it might be necessary to shift the whole project state by a "brute-force" manipulation of state variables instead of performing backtracking. The problem is how to modify the models according to the new state of the real-world project and to preserve existing dependencies.

P2: *Type-Instance Correspondence.* Direct manipulation of instances results in a project trace which cannot be described by the plan's process types. This is also true when types of active processes are modified. Mechanisms are needed to establish correspondence between instances and types.

P3: *Existing Products.* Backtracking and choosing an alternative should not mean throwing away the results already produced. Mechanisms for reuse within a project are needed.

Although the requirements listed in this section present a unified view of both approaches this does not mean that a particular user has to manage such a tool in its entire com-

Published in: Proceedings of the 8th International Conference on Software Engineering and Knowledge Engineering, June 10-12 1996, Lake Tahoe, Nevada, USA

plexity. Some features might be interesting for only some roles (e.g., a project planner is interested in R4 but not in R9) and some functionality should be kept completely away from users (e.g., R10).

6 Related Work

To validate the completeness of our integration we compared it against existing frameworks and definitions that were developed by Conradi, Fernström and Fuggetta [8], Feiler and Humphrey [11], Lonchamp [16], and Armitage and Kellner [2] (see Table A.1 of the appendix). Every framework provides a consistent set of concepts that embodies a particular understanding about aspects of software processes. In contrast to the concepts presented in this paper, aspects of the meta-process (i.e., the process of process modeling) are also described in some of the works [8, 11, 16]. The four definition frameworks are not formalized but natural language is used to explain the meaning of concepts. Because the terms were developed in different contexts, one cannot assume a perfect match between them. Therefore, we see the terms from different frameworks as similar, not as equal. Under this assumption, CoMo-Kit and MVP-E implement most of the concepts covered by the four frameworks. A predefined type classifying all components of the delivered product (i.e., as proposed by Lonchamp) are not present in either approach. All other abstract concepts covered by the frameworks are considered in MVP-L.

Process-sensitive software engineering environments which support evolution of enacted process models are a focal point of current research, but the results are still immature [19, 27]. In the remainder of this section we discuss environments relevant for the work presented in this paper and point out the main differences to our approach. Important requirements not met by the related approaches are checked (which leaves open whether the other requirements are met). The unsatisfied requirements are marked by a '¬'.

The SPADE environment is a system for developing analyzing, and enacting process models described in the language SLANG (Spade LANGuage) [3]. *Activities* are modules with well-defined interfaces and a Petri net specification as a body. Activity types may be changed during enaction but they do not affect existing instances. When the type of an active process is modified, SPADE prompts the user to provide a transformation function. This is a solution for the problem P2 (i.e., type-state correspondence) which is not solved by our approach. SLANG provides only a small set of software development process concepts (¬R1). Also the user must decide when to start process evolution. The system does not provide any support to decide which parts need to be changed (¬R5).

GRAPPLE is an operator-based approach which supports planning and plan recognition [13]. The operators encapsulate the functionality of both tools and processes performed by agents. Reason maintenance techniques are used to manage dependencies between process steps; dependencies between products are not maintained (¬R5). Our synthesized approach extends these techniques by explicitly representing dependencies between products, so that a goal-directed reaction on changes of the product state is possible. GRAPPLE does not allow for alternating planning and enactment (¬R4).

The database-oriented EPOS Process Modeling System distinguishes between classes (templates), instances thereof, and information about the creation, change, and conversion of classes and instances on a meta-level [14]. Feedback about correctness and performance of the enacted process model triggers changes of classes and instances which are under version control. Classes and instances may be changed in the case of inactive processes. The user is responsible for establishing consistency between classes and instances. Thus the EPOS Process Modeling System tackles the problems P1 and P2. No dependencies between process fragments are managed (¬R5) so it is not possible to determine what processes accessed a faulty product and might be enacted another time. Detection of deviations and recognition of a change's impact are completely left to the user (¬R6).

Redoing is an operation in the Hierarchical and Functional Software Process (HFSP) approach that allows cancellation of erroneous activities and doing that part of the process again [28]. Software development processes are understood as functions organized in a hierarchy (called an enaction tree). Redoing means cutting a subtree out of the enaction tree and replacing it with another tree which is newly enacted. The decision to redo is specified in the process models. It can be seen as a sort of "goto" where results in the subtree are discarded. In contrast to our approach, short-term planning is not supported (¬R3), the process models must be completely defined before interpretation (¬R4), and the decisions for redoing are predefined, which means that criteria to detect deviations from the plan must be specified within the models (¬R5).

7 Summary

Supporting real-world processes requires an understanding of their concepts. Process support systems that manage their own process representations need special mechanisms to keep track about what is going on in the real world. Sophisticated mechanisms should support even complicated processes (e.g., software development processes). Many process support systems were developed which provide single solutions for specific problems. A next-genera-

tion process-sensitive software engineering environment should not be built from scratch. Existing environments already provide a basis towards powerful environments, although a lot of problems are still unsolved.

This paper presents the first steps of synthesizing two approaches, namely CoMo-Kit and MVP-E. They both were developed completely independently to solve particular and isolated problems of automated process support. Assumptions were made during their evolution which are the boundaries for their application. The synthesis of CoMo-Kit and MVP-E extends the number of potential process scenarios to be supported. We discussed the correspondences and differences of both approaches in order to derive a set of requirements which present a unified view on process-sensitive software engineering environments. This discussion helps to understand what is intrinsic to processes in general and what is special to software development processes, by highlighting the correspondences and differences of MVP-E and CoMo-Kit.

Roughly spoken, as an intermediate result of the synthesis MVP-L's concepts are used to describe software processes and the concepts of the CoMo-Kit process engine are used to enact the models. Using these basic elements we are able alternating between modeling, planning, and enaction modes. Our integrated approach provides a sufficient set of concepts to capture real-world processes. By integrating knowledge engineering techniques a flexible process-sensitive software engineering environment will be created which manages dependencies between project information and supports backtracking.

Acknowledgements: Sigrid Goldmann did careful work comparing CoMo-Kit and MVP-L. Chris Lott gave valuable feedback on an earlier version of this paper. The authors would like to thank the members of both working groups who contributed to many ideas presented in this paper.

References

[1] P. Armenise, S. Bandinelli, C. Ghezzi, and A. Morzenti. Software process languages: Survey and assessment. In *Proceedings of the Fourth Conference on Software Engineering and Knowledge Engineering*, Capri, Italy, June 1992.

[2] James W. Armitage and Marc I. Kellner. A conceptual schema for process definitions and models. In Dewayne E. Perry, editor, *Proceedings of the Third International Conference on the Software Process*, pages 153–165. IEEE Computer Society Press, October 1994.

[3] Sergio C. Bandinelli, Alfonso Fuggetta, and Carlo Ghezzi. Software process model evolution in the SPADE environment. *IEEE Transactions on Software Engineering*, 19(12):1128–1144, December 1993.

[4] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Goal Question Metric Paradigm. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 528–532. John Wiley & Sons, 1994.

[5] Victor R. Basili and H. Dieter Rombach. The TAME Project: Towards improvement–oriented software environments. *IEEE Transactions on Software Engineering*, SE-14(6):758–773, June 1988.

[6] Israel Z. Ben-Shaul, Gail E. Kaiser, and George T. Heineman. An architecture for multi-user software development environments. In H. Weber, editor, *Proceedings of the Fifth ACM SIGSOFT/SIGPLAN Symposium on Software Development Environments*, pages 149–158, 1992.

[7] Alfred Bröckers, Christopher M. Lott, H. Dieter Rombach, and Martin Verlage. MVP–L language report version 2. Technical Report 265/95, Department of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, 1995.

[8] Reidar Conradi, Christer Fernström, and Alfonso Fuggetta. A conceptual framework for evolving software processes. *ACM SIGSOFT Software Engineering Notes*, 18(4):26–35, October 1993.

[9] Bill Curtis, Marc I. Kellner, and Jim Over. Process modeling. *Communications of the ACM*, 35(9):75–90, September 1992.

[10] Barbara Dellen, Kirstin Kohler, and Frank Maurer. Design rationales and software process models. SFB-Bericht SFB501-01-95, Fachbereich Informatik, Universität Kaiserslautern, 67653 Kaiserslautern, November 1995.

[11] Peter H. Feiler and Watts S. Humphrey. Software process development and enactment: Concepts and definitions. In *Proceedings of the 2nd International Conference on the Software Process*, pages 28–40. IEEE Computer Society Press, February 1993.

[12] D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed & Parallel Databases*, 3:119–153, 1995. Kluwer Academic Press, Boston.

[13] Karen Erickson Huff. *Plan-Based Intelligent Assistance: An Approach to Support the Software Development Process*. PhD thesis, University of Massachusetts, September 1989.

[14] M. Letizia Jaccheri and Reidar Conradi. Techniques for process model evolution in EPOS. *IEEE Transactions on Software Engineering*, 19(12):1145–1156, December 1993.

[15] C. D. Klingler, M. Neviaser, A. Marmor-Squires, C. M. Lott, and H. D. Rombach. A case study in process representation using MVP–L. In *Proceedings of the 7th Annual Conference on Computer Assurance (COMPASS 92)*, pages 137–146, June 1992.

[16] Jaques Lonchamp. A structured conceptual and terminological framework for software process engineering. In *Proceedings of the Second International Conference on the Software Process*, pages 41–53. IEEE Computer Society Press, February 1993.

[17] Christopher M. Lott. Measurement support in software engineering environments. *International Journal of Software Engineering & Knowledge Engineering*, 4(3):409–426, September 1994.

[18] Christopher M. Lott, Barbara Hoisl, and H. Dieter Rombach. The use of roles and measurement to enact project plans in MVP-S. In W. Schäfer, editor, *Proceedings of the Fourth European Workshop on Software Process Technology*, pages 30–48, Noordwijkerhout, The Netherlands, April 1995. Lecture Notes in Computer Science Nr. 913, Springer–Verlag.

[19] Nazim H. Madhavji and Maria H. Penedo. Guest editor's introduction. *IEEE Transactions on Software Engineering*, 19(12):1125–1127, December 1993. Special Section on the Evolution of Software Processes.

[20] Frank Maurer. *Hypermedia-based Knowledge Engineering for distributed, knowledge-based Systems*. PhD thesis, Universität Kaiserslautern, 1993. in German.

[21] Frank Maurer and Jürgen Paulokat. Operationalizing conceptual models based on a model of dependencies. In A. Cohn, editor, *Proceedings of the 11th European Conference on Artificial Intelligence*. John Wiley &

Published in: Proceedings of the 8th International Conference on Software Engineering and Knowledge Engineering, June 10-12 1996, Lake Tahoe, Nevada, USA

Sons, Ltd., 1994.

[22] Andreas Oberweis. Workflow management in software engineering projects. In S. Medhat, editor, *Proceedings of the 2nd International Conference on Concurrent Engineering and Electronic Design Automation*, 1994.

[23] Ch. Petrie. *Planning and Replanning with Reason Maintenance*. PhD thesis, University of Texas, Austin, 1991.

[24] Charles Petrie. Context maintenance. In *Proceedings of the AAAI–91*, 1991.

[25] H. Dieter Rombach and Martin Verlage. How to assess a software process modeling formalism from a project member's point of view. In *Proceedings of the Second International Conference on the Software Process*, pages 147–158, February 1993.

[26] H. Dieter Rombach and Martin Verlage. Directions in software process research. In Marvin V. Zelkowitz, editor, *Advances in Computers, vol. 41*, pages 1–63. Academic Press, 1995.

[27] Stanley M. Sutton, Jr., Dennis Heimbigner, and Leon J. Osterweil. Language constructs for managing change in process–centered environments. In *Proceedings of the 4th ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments*, pages 206–217, 1990.

[28] Masato Suzuki, Atsushi Iwai, and Takuya Katayama. A formal model of re–execution in software process. In Leon J. Osterweil, editor, *Proceedings of the 2nd International Conference on the Software Process*, pages 84–99. IEEE, IEEE CS Press, February 1993.

[29] Martin Verlage. Multi–view modeling of software processes. In Brian C. Warboys, editor, *Proceedings of the Third European Workshop on Software Process Technology*, pages 123–127, Grenoble, France, 1994. Nr. 772, Springer–Verlag.

Appendix

Table A.1 relates software process terms defined by different authors. The terms explain real-world concepts. The definitions given in [2, 8, 11, 16] are in natural language and therefore lack formality. The overview presented in Table A.1 should not be understood as a precise comparison of terminology. The terms differ slightly in their meanings even when they have the same name. The matching was performed based on careful but subjective assessment. The reader is referred to the cited literature for a detailed explanation of the terms. A table cell two high means that the term corresponds to two terms of another framework. Empty cells mean that no term with an equivalent meaning to other terms of that row is defined in the approach discussed in the column.

CoMo-Kit	MVP-L	Armitage and Kellner [2]	Feiler, Humphrey [11][a]	Lonchamp [16][a]	Conradi et al. [8][a]
Task	Process	Process	Process (Element)	Software Process	Process
					Production Process
Method		Activity		Process Step	Activity
		Activity Description	Process Script		
		Procedure	Process Program		
	Elementary Process[b]		Process Step	Activity	
			Task	Task	
	Process Attribute	Activity State[c]			
	Refinement	Decomposition			
	Criteria	Behavioral Information	Process Constraint	Constraint	
	Project Plan		Process Plan		
			Project Plan		Software Project
Concept	Product	Artifact		Artifact	
	Elementary Product				Software Item
					Software Product
				Deliverable	
	Product Attribute	Artifact Statec			
	Product Flow	Artifact Flow			
	Resource			Resource	
Agent	Personnel	Agent	Agent	Agent	Agent
	Tool				Tool
	Resource Attribute	Agent Statec			
	Attribute	Attribute[d]			
Type	Model	Process Definition	Process Definition	Generic Process Model	Template

Table A.1: Comparative List of Process Concepts

a. Not all process terms presented in [8, 11, 16] are considered in the corresponding columns, because many of them describe no concept of software development processes but ideas of enaction (e.g., enactment state), organizational processes (e.g., monitoring), or process characteristics (e.g., liveness).

b. Process which contains no refinement.

c. Predefined attributes used by a process engine (interpretation machine) to manage an overall project state.

d. Attribute is explained as "a textual description of information". This general and abstract definition matches the other terms in the row only partially.