Quality Information Systems:
Repository Support for Evolving Process Models

Klaus Pohl and Matthias Jarke[1]
Informatik V, RWTH Aachen, Ahornstr. 55,
5100 Aachen, Germany
{pohl, jarke}@informatik.rwth-aachen.de

Abstract. *Relationships between Total Quality Management and process support in CASE environments are established in two ways: firstly, by analysing the repository requirements for each stage in the SEI process maturity model, enhanced by team support aspects; secondly, by presenting a quality-centered process model that formally differentiates but also integrates the aspects of process definition, process evaluation, and process improvement.*

---

[1] This work was supported in part by ESPRIT Basic Research Action 6353 (NATURE) which is concerned with Novel Approaches to the Theory Underlying Requirements Engineering and by the state Nordrhein-Westfalen, Germany.

1 Introduction

Quality Assurance and Quality Control are common methods for ensuring the quality of a product (not only for software products). *Quality Assurance* is defined in ISO 8402 as the aspect of the overall management function that determines and implements the *quality policy*; where *quality policy* is defined as the overall quality intentions and objectives of an organization as formally expressed by senior management. *Quality Control* is defined in ISO 9001 as the operational techniques and activities that are used to satisfy the quality.

Early work on quality contol and assurance has mostly focused on the product, evaluated through testing or metrics. While these are undoubtedly valuable contributions, industrial experiences indicate that quality cannot be tested after the fact but must be produced. This observation emphasizes the process rather than the product. Process orientation and integration, teamwork support, and simultaneus engineering of multiple aspects (requirements, design, implementation and maintenance considerations) across tool boundaries are among the more glaring problems not adequately addressed in current design environments.

Total Quality Management (TQM) is a philosophy popularized through the influential works of Deming which have been summarized in his famous 14 points (Deming 1986). TQM takes a radical stance on process versus product orientation. The enterprise is seen as a process whose only constant purpose is continuous improvement of process quality (which will automatically lead to product quality). Specific techniques such as Quality Function Deployment (Hauser and Clausing 1988) emphasize the role of the customer as the final judge of quality. Purely quantitative output productivity measures as well as mass inspection (e.g., intensive product testing in software engineering) should be given up in favor of conitinuous search for process improvements, teamwork, and statistical process control.

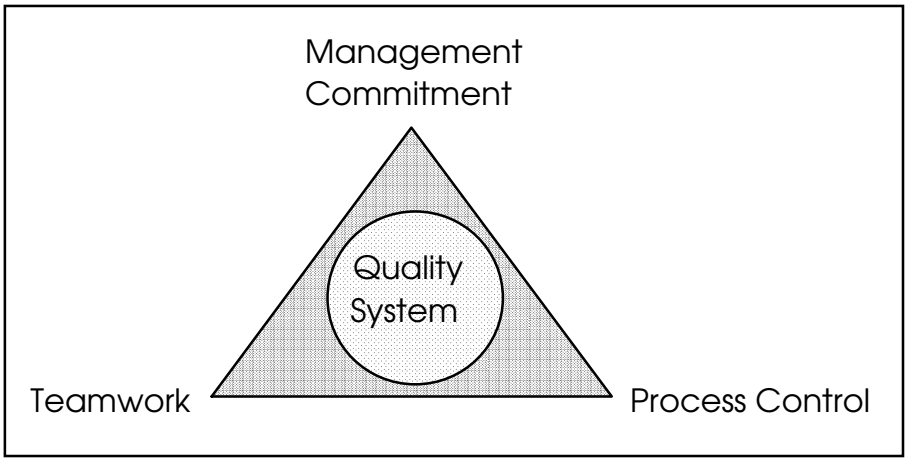

Fig. 1: The TQM triangle (adapted from Oakland 1989, p. 15).

Figure 1 summarizes the TQM cycle. Management commitment towards global quality orientation is supported by selective process control. These process controls deliver raw data from which teamwork can derive concrete proposals for process improvement; management

commitment has again to ensure that teamwork results are actually instituted and not lost through bureaucratic hurdles. An important prerequisite for this to work is to "drive out fear" (Deming´s rule no. 8). Punishing individuals for errors will cause them to suppress rather than to suggest improvements.

The domain of software production and maintenance has recently been concerned with the development of formal process models as a basis for guiding and documenting the software process. Formalisms have been derived from distributed artificial intelligence (Dowson 1987), object-oriented knowledge representation schemes (Jarke, Jeusfeld and Rose 1990), Petri nets (Madhavji et al. 1990), and other areas. However, with the exception of cleanroom development (Selby, Basili, and Baker 1987), few of them have been tested in truly large-scale development environments.

One of the reasons may be that introducing improvement-orientation is a major organizational change. One important question -- how a company can go about this -- has been addressed by the software process maturity and change model proposed by the Software Engineering Institute (Humphrey 1990). Humphrey defines six steps of any improvement cycle for software capabilities:

1. understand the current status of your development process
2. develop a version of the desired process
3. establish a list of required process improvement actions in order of priority
4. produce a plan to accomplish the required actions
5. Commit the resources to execute the plan
6. start over at step 1.

He also suggests standard maturity levels on which the analysis of the current situation and the definition of the next step could be built. His five levels (shown in table 1) have meanwhile been widely adopted as an evaluation standard for software suppliers. With suitable adaptations, they also appears useful for the categorization of other service-oriented processes.

Process level	Characteristics	Needed Actions
initial	Chaotic - unpredictable costs, schedule and quality performance	Planning (size and cost estimates and schedules), performance tracking, change control, commitment management, Quality Assurance
repeatable	Intuitive - cost and quality highly variable, reasonable control of schedules, informal and ad hoc process methods and procedures	Develop process standards and definitions, assign process resources, establish methods (requirements, design, inspection and test)
defined	Qualitative - reliable costs and schedules, improving but unpredictable quality performance	Establish process measurements and quantitative quality goals, plans, measurements and tracking
managed	Quantitative - reasonable statistical control over product quality	Quantitative productivity plans and tracking, instrumented process environment, economically justified technology investments
optimizing	Quantitative basis for continued capital investment in process automation and improvment	Continued emphasis on process measurement and process methods for error prevention

Table 1: Process levels characteristics and needed actions (Humphrey 1990, p.56)

If we compare TQM with the process maturity model, the latter can be understood as a structure to organize the implementation of TQM in the software domain. However, the emphasis is slightly different. Humphrey and many related works in the software process (Osterweil 1987, Madhavji et al. 1990) and software quality domain (Basili and Rombach 1988) focus on imposing structure and measurement on the process. They, perhaps unwillingly, create the impression that quality orientation can be achieved by central management alone. In contrast, writers such as Deming and Oakland emphasize open sanction-free teamwork and actually discourage quantitative goals in favor of worker motivation.

Thus, the Process Maturity and TQM models seem to exhibit complementary strengths and weaknesses. A full improvement-oriented environment should feature both, precise process understanding and management as in the SEI model, and the teamwork orientation of TQM.

Achieving this goal poses a host of organizational challenges but also some important technical questions. This paper investigates the role of repository technology in satisfying the requirements induced by the TQM philosophy. After a brief example that illustrates the shortcomings of current design environments with respect to the SEI model (section 2), we first present the concept of process-oriented repositories as an extension of the well-known Information Resource Dictionary Standard (IRDS 1988). This allows us the means to characterize more closely the kind of support needed at each stage of the SEI model (section 3). We call such extended repositories *quality information systems*.

The main contribution of this paper is in section 4 which presents a comprehensive process model for quality information systems, centering around the explicit representation and management of quality goals. The power of this approach is demonstrated by showing how it integrates several recent proposals for situation-based process modeling (Grosz and Rolland 1991), goal-oriented process programming (Mylopoulos et al. 1992), goal-oriented measure–ment (Oivo and Basili 1992), and process change management (Madhavji 1992). Section 5 gives preliminary evidence of substantial research and industry experiments being carried out with the model in different areas of software quality management as well as in industrial engineering.

2. An Example from Information Systems Engineering

In this section, we use a simple example from information systems development to show how traditional CASE fails in a chaotic process situation[2] .

---

[2] A more detailed version of the example, including models for dealing with the problems mentioned here, can be found in (Jarke and Pohl 1992).

A group of system developers and other stakeholders have used entity-relationship modeling (ER, Chen 1976) and structured analysis (SA, DeMarco 1978) to define the requirements for a personnel information system. The resulting conceptual data model — summarized in the ER part of figure 2 — involves entities such as departments, projects and people, relationships such as assignment of people to projects, and an integrity constraint that employees should only work on projects of their own department. On the process side, system analysts and users have defined two kinds of activities (SA bubbles) for hiring employees. One function is intended for hiring an employee for a department, the other is intended for hiring an employee for a specific project. In object-oriented terminology, the second function is a specialization of the first one.

Suppose there is a CASE tool which can convert each bubble in such an SA diagram automatically into a transaction specification on a database (which, in turn, was generated from the ER part of the diagram). This transaction specification is then realized by a forms interface in which the user can enter input parameters and see output data.

In our example, this results in the mapping shown in the lower part of figure 2; the system works and is delivered to the customer.

Surprisingly for the proud CASE tool owner, the customer is not happy with the result. Users complain that it is too complicated to switch between the different windows. Being the ingenuous programmer he is, the developer looks up a good book on conceptual modeling and comes up with the idea that it should be sufficient to define only one transaction specification for the most special case, here: the hiring of employees for projects. People just hired for the department in general are simply given some artificial project number. He changes the code by simply dropping the left of the two transactions in fig. 2 and re-delivers the code to the customer who is happy with this solution (at least for the moment).

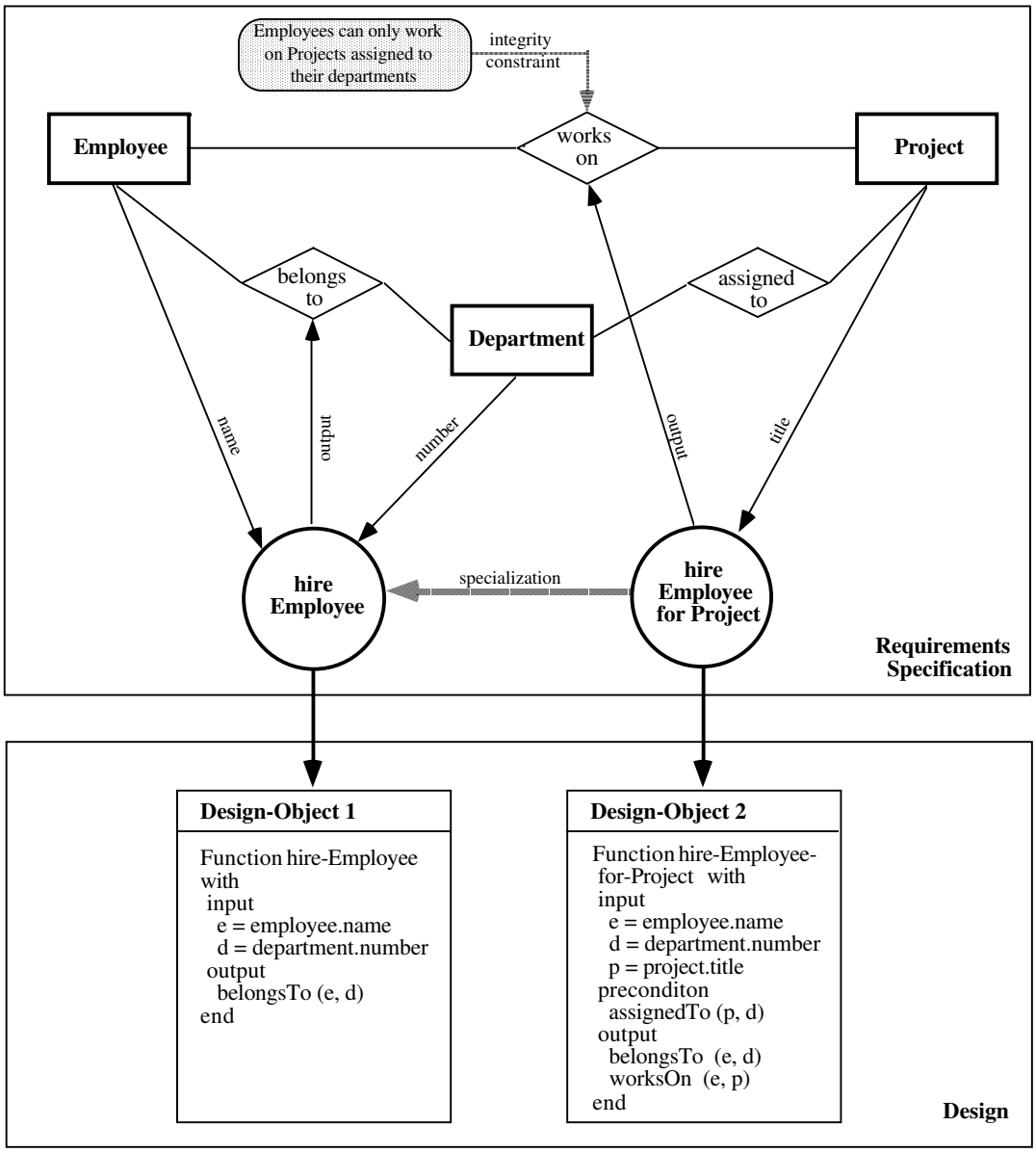

Fig. 2: Design generated from the requirements by the CASE tool

The salesperson of the CASE company would tell you that this is the way to go. The CASE tool makes a proposal, if you don´t like it, simply change the results. What´s wrong with it?

Unfortunately, from the viewpoint of TQM and SEI, the answer is: *just about everything!*

(1) The manual change of tool output leaves no visible *trace* in the system, except maybe a file version number.

(2) The fact that a specific well-known *method* for mapping of isA hierarchies has been used is nowhere documented.

These two problems alone make the process unrepeatable. If a further error were detected in the requirements model, its correction could not take advantage of the existing implementation for propagating the change. But there is more:

(3) The *reason* why the second method was chosen over the one of the CASE tool is known only to the individual who did the work. Even if the method were available in an extended tool, others would not know why one is better for this case than the other. This lack of quality-oriented documentation impedes teamwork and quality awareness.

(4) Quality is only improved by (involuntary) testing by the customer, not produced in the first place because there is no mechanism to generalize experiences like this into rules that prevent similar errors in the future. The same problem may occur again and again in the organization. A *process definition* is lacking which could contain a regulation to prefer the second method for reasons of user-friendliness.

(5) It is unclear whether the particular argument used by the programmer in our case is the only one that impacts user friendliness — it was just the breakdown that happened to occur. The choice may depend on several factors with trade-offs dependent on the organization. Thus, the representation of process definitions must take into account that regulations are *organization-specific*, dependent on quality factors, and subject to frequent change. In fact, this continuous improvement orientation is the very essence of TQM.

(6) Last not least, it is highly unlikely that all improvement ideas will originate from the same person. An organizational and technical environment must be created in which quality-oriented *teamwork* not only within the development team but also with other stakeholders is actively supported. Version and configuration control is needed to organize this process.

The above problem list contains a lot of organizational measures that may be independent of tools. However, it does not indicate a single aspect where the CASE tools contribute positively to these measures. The only exception is repeatability of the very function the tool is hardcoded to support. But even that is only true if the tool does not contain an opaque collection of methods where the environment has no means to observe the process from outside.

There are two conclusions to be drawn from this analysis. On the one hand, one can derive a checklist of factors which make a CASE tool suitable for a quality-oriented environment. Such a checklist might include goals such as:

- clear definition of the methods used
- coverage with respect to all important methods known for the task at hand
- extensibility to adopt newly invented methods
- understandability as a basis for team communication
- traceability of process choices inside the tool and communicatability to the repository
- adaptability of the process specification in the tool.

This list should be just considered indicative of things to look for. Miyoshi et al. (1989) report a major experiment on the evaluation of CASE environments which contains a more comprehensive list of features identified by a group of Japanese experts.

On the other hand, TQM cannot rely on individual CASE tools alone but must address the whole product lifecycle, from requirements capture to the maintenance phase, if only from the viewpoint of getting better ideas for new products. While there are a host of

organizational methods known from other TQM applications, the only serious candidates for *technological* TQM support seem to be the communication, collaboration, and coordination facilities recently promoted under the label of repositories or design information systems.

3 Repository Management and Process Maturity

A repository is the central information system around which design environments are layered. Efforts such as the European PCTE initiative (Thomas 1988) and IBM´s AD/Cycle repository manager (Sagawa 1990) have created substantial interest in this technology. While initial repositories were mostly oriented towards the representation of objects and their relationships, more recent proposals emphasize process dependencies and even project management aspects.

Our approach starts from the basic repository classification hierarchy defined in the Information Resource Dictionary Standard IRDS (1988) and shown in fig. 3 (Smolander et. al. 1991). The model consists of four levels where level *n+1* formally constitutes a type system for level *n* (*n*=1, 2, 3). The lowest level, the *production data* layer, consists of actual databases and maybe software execution traces; it is not directly part of the repository but often maintained by the individual subsystems coordinated by the repository. The second level, the *data* layer, describes and relates schemata (data and procedures) developed for each subsystem application; this constitutes the data layer of the software database, i.e., the repository. The third layer is the IRD *schema* layer which represents, in a uniform framework, the individual data and application languages in which the various subsystems are defined; thus, it defines the schema of the repository. Finally, the uniform descriptive formalism used to define the schemata is defined in the fourth level, the *schema description* layer.

The first observation to be made when addressing quality or quality improvement through repository technology is that all layers have to become process oriented rather than just product oriented. This is helped by the observation that the repository as shown in the middle of fig. 3 misses the lowest layer. As a consequence, the "model of application systems" layer and everything above it can contain — in addition to the models of subsystems — concepts that are not physically in the computer, including recording of the process.

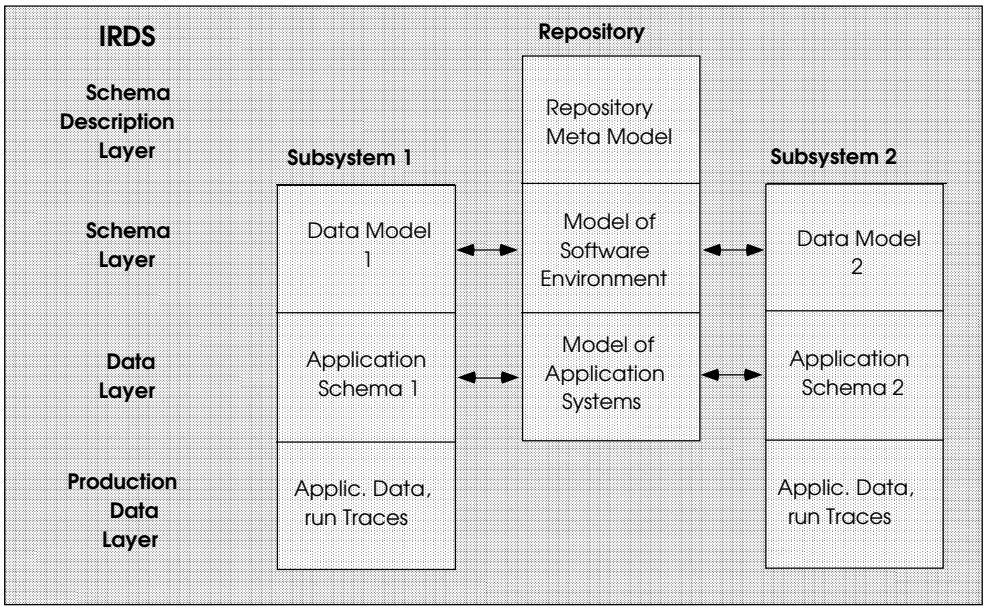

Fig. 3: Repositories for integrating software objects

Fig. 4 shows how, consequently, a process oriented repository hierarchy would look like. The lowest level would still just contain application data and product execution traces but these are now interpreted as tests of product quality which can be used for feedback of process execution. The second layer records process executions. If a formal process definition at the third level does not exist, recording process can still serve a number of useful purposes, such as recording the rationale for design decisions for future reuse. If a formal process definition does exist, recording its execution may serve as a basis for measurement of product and process, checking of correct enacting of process, and feedback to the third level concerning the quality of the process definition.

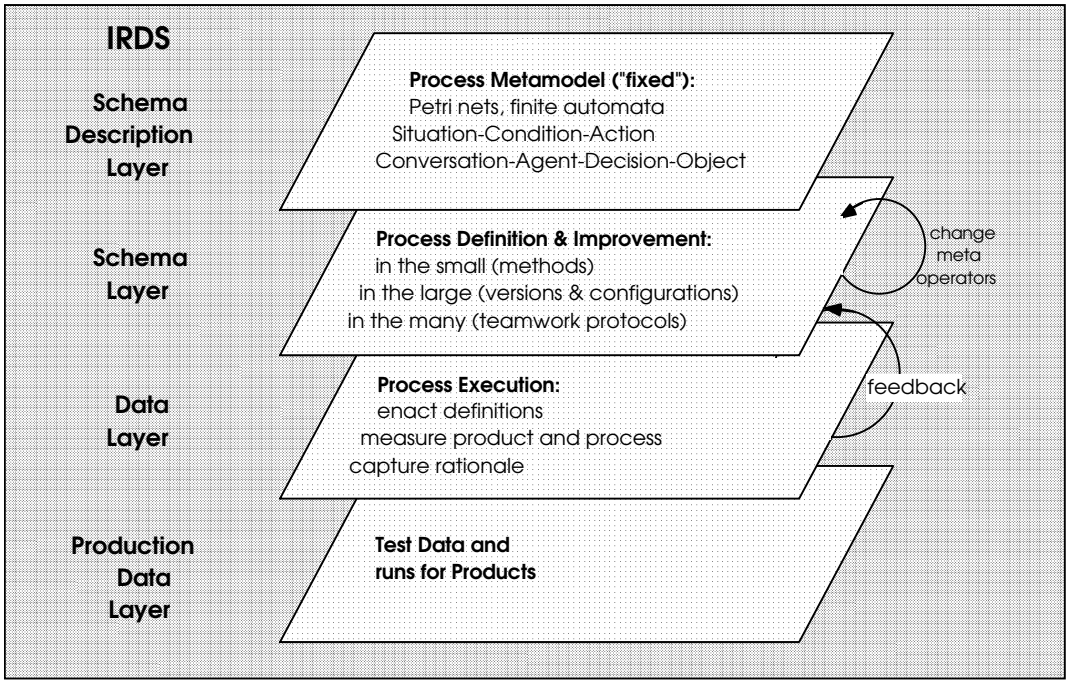

Fig. 4: Repositories for process definition and evolution

The third level contains process definitions. Such definitions can be organized according to various kinds of abstraction hierarchies or viewpoints. In our work, we have found it useful to distinguish the viewpoints of development in the small (content oriented methods for specific individual tasks), in the large (version and configuration management from both object and task viewpoints), and in the many (team cooperation in the preparation, execution, and evaluation of processes). Others have considered hierarchies according to the scope and precision of process definitions, such as individual, group, organization, city, state, and so forth (Perry and Kaiser 1991, Madhavji 1992).

A process oriented repository meta model at the fourth level provides the basic ontological structure according to which processes are defined.

Comparing the requirements for quality support listed in section 2, and the repository structure of fig. 4, we are now in the position to associate some fairly precise requirements with each of the stages of the SEI maturity model of fig. 2. The result is shown in table 2.

For repeatability, the repository must be able to model tools and methods withing them. Application of these tools and methods creates dependencies which must be recorded together with their rationales. In terms of fig. 4, all of this can be done at the second (data) layer of the

repository hierarchy, governed by a fairly simple type structure at the schema layer. Simple database technology (without metamodeling facilities) can do this.

To support the stage of defined or managed processes, it must be possible to define and change process schemas; therefore, we have to introduce the meta level in which a schema description structure for process and product can be defined. Table 2 lists features of what needs to be done to address the problems highlighted in the example; perhaps most interesting is the need for being able to control different methods within and between complex tools so that the process definition can actually be enacted.

The last step shown in table 2 is the task of process definition management and optimization. The choice of a good schema description structure is very important for what kinds of process definitions and changes thereof can be expressed. Based on the general objectives of TQM, table 2 emphasizes aspects of process evaluation, teamwork, and goal-oriented process re-definition.

process maturity levels	repository concepts	problems addressed	requirements on CA
repeatable at project (instance) level	tools and methods dependencies decisions and rationales	traceability of processe repeatable operations	coverage of life-cycle embedding in repositor visibility of (intermedia
defined at environment (class) level	product documentation specification of languages, methods and tools quality oriented process models quality control and assurance models organization models version and configuration models	producing quality standardization of proce organization-specific de reference models)	external control of method choices synchronized use of too repository
managed/optimized according to TQM philosophy	process evalutation models goal-metric relationships process presentation models teamwork models process improvement operators	measurement & feedbac teamwork support process evolution	exchangeable methods and composable tools multi-user interface open for measurements

Table 2: Tool and repository requirements for process maturity stages

As indicated in fig. 4, a number of proposals have been advanced for process meta models. One group considers the "programming" of processes from a centralized perspective and offers program control structures, finite automata, or Petri nets as process definition languages (Osterweil 1987, Madhavji et al. 1990). Others take a more "opportunistic" attitude and define a process in terms of pre-defined (re-) actions on given situations (Kaiser et al. 1988, Grosz and Rolland 1991). Yet others emphasize the teamwork nature of process definitions and the role of human decisions (Potts and Bruns 1988, Hahn et al. 1991, Rose et al. 1991).

If we focus on introducing improvement orientation into the repository, fig. 4 demonstrates that — regardless of the specific formalism selected — a suitable process meta model must be able to relate at least three kinds of actions to each other in a meaningful way (fig. 5):

- product actions associated with the actual product *process definition.*

- control or monitoring actions, providing *feedback* from process execution to process definition.
- improvement actions that provide a structured way to change process definitions (*change meta operators*).

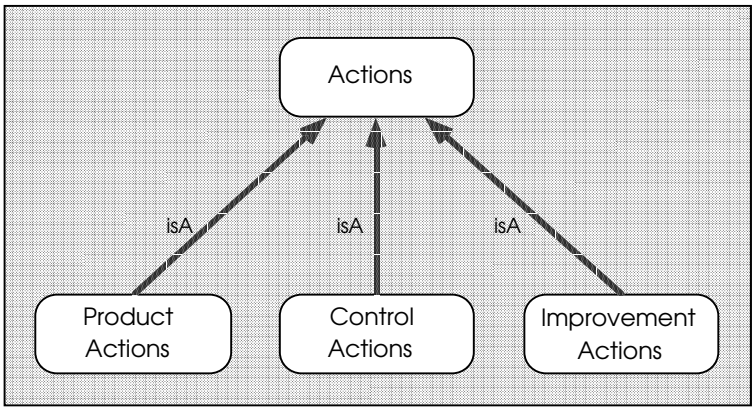

Fig. 5: Action types for improvement-oriented repositories

The main goal of our model will be to elaborate and interrelate these three kinds of action. Our strategy to achieve this goal is based on the observation that, at least the latter two kinds only make sense when they are based on explicit modeling of quality goals: we have to say *what* we control, towards what we improve, even what drives design decisions within a product process model. The remainder of this section presents our approach to organizing quality goals.

A quality-oriented process metamodel achieves its potential only when instantiated with detailed knowledge about specific quality factors and methods for their achievement. This domain analysis has two aspects: definition of an *overall framework* in which goals can be classified, and development of *goal-specific process models* how to reach specific goals with specific product classes.

For the former purpose, we are using the framework sketched in figure 6 which was originally proposed for the special case of information systems (Jarke 1990) but seems to have borader applicability. Quality requirements are organized according the relationships among four "worlds" which are relevant to the system being developed: the *subject world* which is represented by the system with a certain degree of quality; the *usage world* (organizational environment) in which the system functions with a certain degree of reliability, user satisfaction and organizational effectiveness; and the *development world* in which the system technically evolves with certain costs, a certain degree of reusability, and other software quality factors. Finally, the *system world* has its own internal quality factors such as proven correctness with respect to formal specifications, need for internal system resources, and the like.

For the example in section 2, the subject world consists of the employees and projects described in the requirements model, the usage world would comprise the personnel department people who hire employees (and complain about the system), and the development world consisting of the developers and their repository and CASE tools. The system world includes the created transaction specifications and their mapping to actual code. The quality problem addressed in section 2 is a problem of ergonomics, i.e., with the usage

world of the system. However, the ad-hoc solution taken by the developer also creates a quality problem in the development world, lack of maintainability.

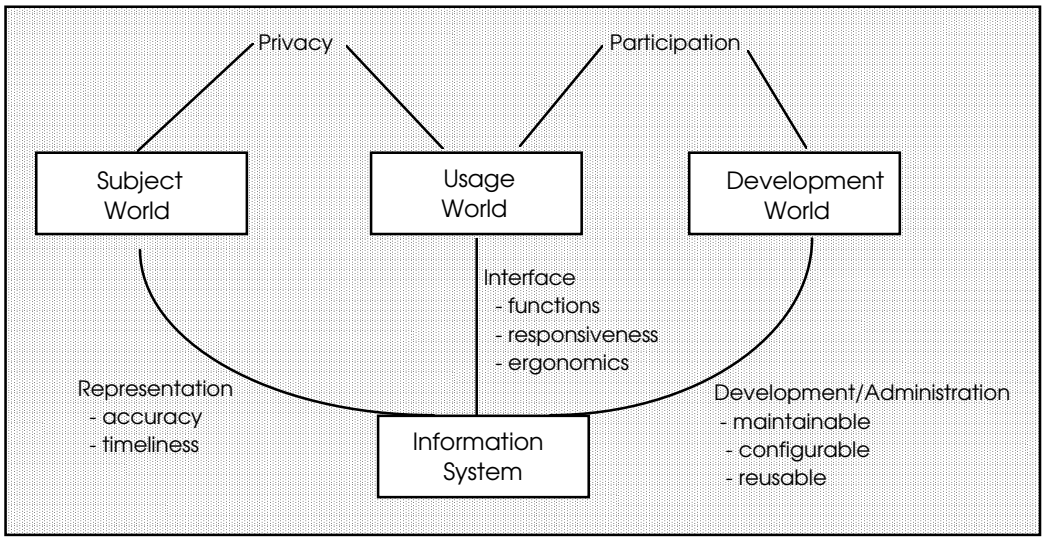

Fig. 6: Organizational framework for information systems quality factors.

There are also direct interactions between the subject, usage, and development worlds which have to be taken into account in process planning and are associated with their own quality factors. The example in section 2 shows obvious interaction between the users and the developers (the complaints), whereas a good process model would have introduced this interaction much earlier through participative design.

What all of this argues for, is to actually represent the stakeholders from the different worlds and their associated goals within a domain-specific quality information systems framework. Similar classifications for quality goals are being attempted in industrial engineering, as a basis for communicating quality information across organizations.

Within such a generic framework, *domain analyses for specific kinds of functional and nonfunctional requirements* create reusable guidelines and repository structures for process evaluation and management. Some expert systems have been developed that support reuse of metrics libraries — e.g., SAMSON (Sirvent and Dupont 1990).

Domain analyses also form the basis for advanced knowedge-based CASE tools. Quite a few projects — including KBSA (Green et al. 1986), ASPIS (Aslett et al. 1989), and DAIDA (Jarke 1990) — have investigated the mapping of functional requirements to designs and code. Concerning nonfunctional requirements, assistants have been designed or implemented for the goals of reliability (Johnson and Malek 1988), algorithm performance (Blaine et al. 1988), information accuracy and database performance (Mylopoulos et al. 1992). Analogous work is going on in the industrial quality engineering area.

Domain analyses do not only apply to methods for individual steps but also to information systems development in-the-large (Rose et al. 1991). Version and configuration management is not only important at the level of individual project histories but also at the metalevel of process improvement. In the example of section 2, the ad-hoc change made manually would be of type (unconstrained) revision, whereas a more principled re-mapping of the requirements using the method from the conceptual modeling book would be characterized more precisely as a variant implementation of the same requirement and would therefore be

much easier to validate and verify. Again, knowledge about the goals (requirements) behind the action of variant creation is the basis for this improvement of maintainability.

4 A Quality-Oriented Meta Model for Processes

Many different formalisms are used for defining processes. A recent comparison of a few alternative approaches (ISPW 1990) includes aspects such as objectives in language design, basic representation formalism, functionality, behavior and so forth — but not the appropriateness for process improvement.

The modeling framework we are going to propose in the sequel is not dependent on any particular formalism, although it does impose certain minimal requirements on formalisms in which it could be computer-supported. Briefly, such requirements include the need for representing multiple meta levels of typing, genericity hierarchies, complex descriptions, and some kind of logic for representing aspects such as policies, laws, or dependency structures. In our practical experiments, we are using the deductive object-oriented database language Telos (Mylopoulos et al. 1990) and its implementation ConceptBase (Jarke 1992). But most aspects of the model can be implemented in any object-oriented database environment, possibly with some more overhead. The graphics we use for showing the models can be understood as direct representations of corresponding views on a Telos knowledge base and thus have a formal semantics.

We elaborate the modeling ideas exposed in the previous section to cover the problems illustrated in the example. First, we discuss in more detail the roles of the different abstraction principles in a process repository model. Next, we look at a simple generic process meta model based on the idea of "situated action" and show how it can be used to categorize other process modeling formalisms. Then, we come back to the distinction and relationships between the three different kinds of actions encountered in a process model, and finally, we discuss the integration of teamwork aspects. The result turns out to be a revision and extension of the CAD^{o} model (Jarke and Rose 1992).

4.1 Abstraction in Process Models

Many researchers recognize the need of distinguishing different levels of abstraction when modeling processes (e.g. Perry 1990, Madhavji 1991, Basili & Rombach 1988). However, they usually focus on one dimension of abstraction which, from the IRDS perspective shown in fig. 3, mixes two concerns:

- the *classification* hierarchy of meta levels such as production data, process execution, process definition, and process definition language, where each level constitutes a type system (as in programming languages) for the level below (cf. introduction)
- the *genericity* hierarchy that deals with different modeling scopes such as organizational dimension (e.g., individual, group, department, organization, inter-organizational, governmental, international) or product dimension (e.g., general software, information systems, knowledge-based information systems, ...). Each level within the genericity hierarchy constitutes a specialization of a level above and therefore usually has to respect its rules, but can change those rules through some feedback formalism as proposed by Madhavji (1992).

Of course, different scoping dimensions can be used to define a process model within the genericity hierarchy (e.g., organization and product specific process model, cf. fig. 7). At the lowest level of the genericity hierarchy, we have project specific process definitions. The need of tailoring project independent models to project-specific models is widely accepted (Osterweil 1987, Basili & Rombach 1988, Perry 1990, Madhavji 1991). They are the ones actually used during the project.

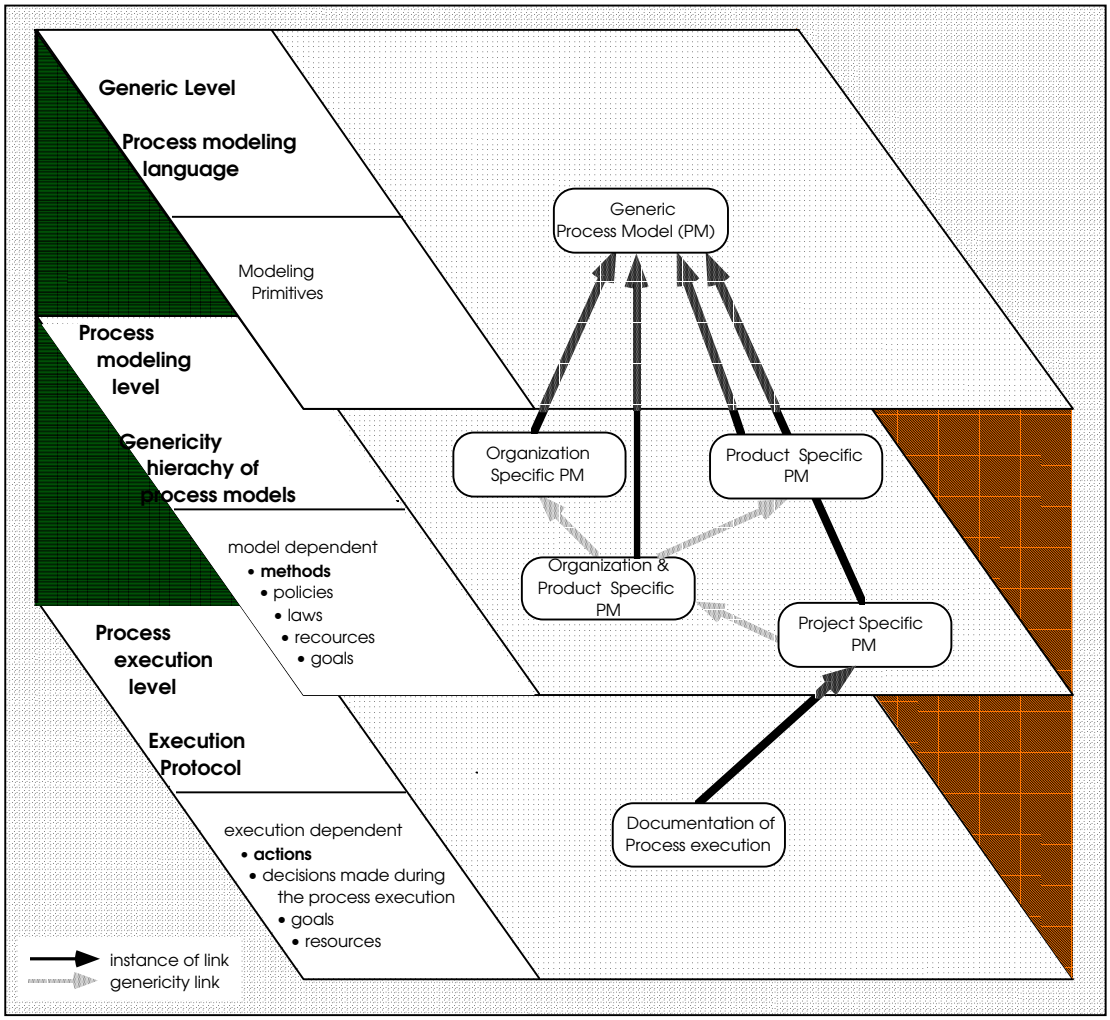

Fig. 7: Classification and genericity hierarchies in process repositories

The distinction between two abstraction hierarchies (classification and genericity) gives a richer picture of the problem-solving strategies that can be adopted in case of process breakdowns. For example, an organization may decide to use ISO 9000 as a policy. It therefore defines overall standard quality assurance methods at the organization-specific process definition level. In addition, product-specific quality assurance methods are defined. If a process model for a special product is built within an organization, both, the organization-specific and the product-specific quality assurance methods must be considered. But also additional methods could be defined for the product & organization specific process model. Using this model further specializiation within the genericity hierarchy (e.g., for building a department model, group model or project model etc.) can be made.

As mentioned, the project-specific process model defines the lowest level of the genericity hierarchy. Should it turn out (e.g., through failed process executions) that certain standard methods selected by the project model are infeasible, there are several choices of navigating within our space of process models. Assume that there are four kinds of process models whithin the genericity hierarchy as shown in fig. 7. Within this framework, the people executing the process have the many choices for correcting process failure. They include:

- revise the process execution within the project process definition (e.g., choose an alternative method that satisfies the same goals).
- delegate the problem upwards to the people defining the project model, to have it solved by someone else within the framework of the organization & product specific process definition.
- change the execution of the project model within the existing organization & process specifc model.
- ask for changes of the organization & product specific model definition (here: the set of standard methods).

In an object-oriented modeling formalism, the genericity hierarchy at the execution level need not be explicitly recorded but is automatically available through inheritance. That is, as shown in fig. 6, we have *one* process execution which is indirectly an instance of each level in the genericity hierarchy to which it is attached. The different genericity levels of process definition have different views which define the process execution at their level of abstraction.

4.2 The Basic Approach: Processes as Situated Actions on Objects

Grosz and Rolland (1991) have proposed the ALECSI process meta model which views a process definition as a generalization of the idea of triggers in active databases (fig. 8). Actions are enabled by certain situations (in contrast to active databases, *not* automatically triggered!). If someone (not shown in the model) decides to "fire" the action, it will take certain objects as inputs and produce or change other objects.

Take the breakdown of a car as an example. If we recognize that the engine of the car is broken, we identify a situation *car engine needs repair* , and may decide to bring the car to the mechanic, if it is a new car. If the car needs a new pink slip within the next month and its overall condition is bad, we don't bring the car to the mechanics. Instead, we identify - through additional information - the situation *car is a wreck,* and apply the action *bring car to the wrecker*. Of course, many situations leave a choice among several possible actions. For example, in the situation *hungry people*, actions such as *go the fridge* or *by food in a shop* or *go to the restaurant* could be applied.

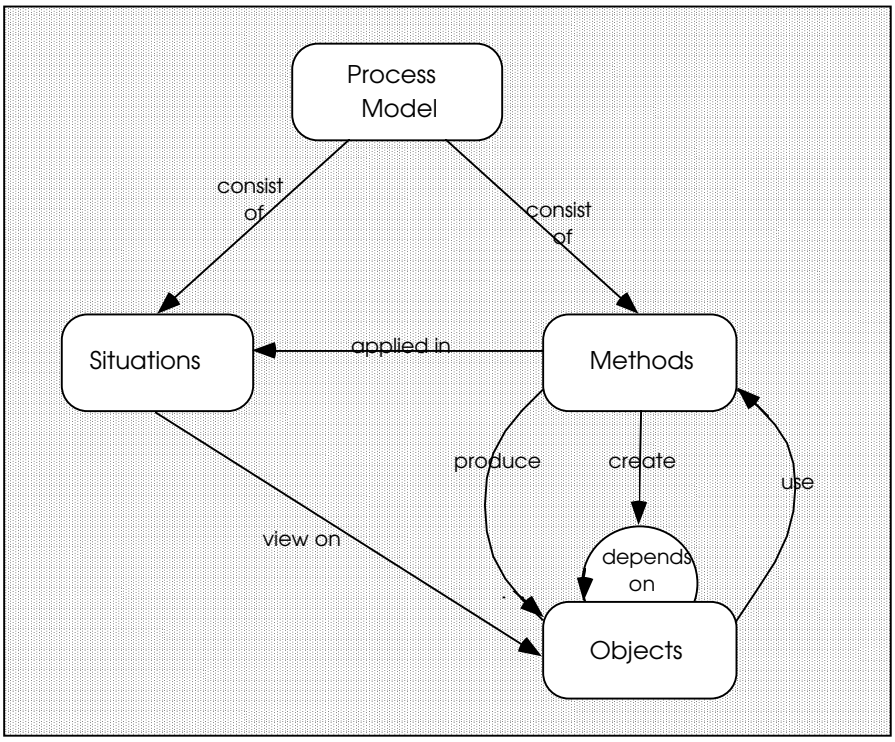

Fig. 8: The kernel of the process meta model

It is interesting to observe that most existing process meta models can be characterized by the ways how they describe situations, actions, and objects in this generic framework:

- Models based on finite automata or simple Petri nets take an uninterpreted "state" as the description of the situation, i.e., the situation is characterized by a (limited) memory of the process history to date; their action part includes — besides making actual progress with the work output — changing this state.
- "Opportunistic" models, including MARVEL (Kaiser et al. 1988) or the DAIDA process model (Jarke et al. 1990) define situations by the state of the object base (which may, of course, contain project status if the user so desires).
- Models based on extended net formalisms (e.g., predicate transition nets) additionally consider the state of the object base through a precondition when deciding on an action. PRISM (Madhavji et al. 1990) is a good example.

In general, an approach looking at process stages as well as the state of one or more objects, appears desirable. The main reason for this is that it does not appear feasible to deduce automatically the situation just given the information in the repository; the explicit claim of a human user or external tool that something has actually been accomplished (or that there is a problem!) is often needed.

However, even then, the example in section 2 demonstrates that a richer picture of the situation is often useful. In particular, we showed the need for representing the rationale of design decisions (underlying the choice of actions, or the situational focus) — shortly, the goals under which to look at situations and actions. The famous example where the optimist sees the glass as half-full, the pessimist as half-empty, also applies here. In section 4.3, we shall demonstrate how goals, and thus explicit quality orientation, can be introduced in all aspects of process modeling.

Since goals do not come from nowhere, this automatically leads to the need for formally integrating teamwork aspects into the model (it is useful to know whether it was the optimist or the pessimist who made the above statement!). We treat this issue in section 4.4.

For all of these reasons, we need to represent the situations themselves as objects in the metamodel rather than just compute them indirectly. A situation is defined by itself, by a view on a restricted set of object states (expressed using a constraint) and by a set of possible actions which could be applied in a particular situation of the defined kind. Additionally, a ranking between the actions is defined. Preconditions defined for each action make sure that it can only be applied within a given (defined) set of situations. The performance of the action can be controlled by the action constraint; e.g., assuring that changes are made to all relevant parts of the products. Postconditions test if, after executing an action, the state of the process (the actual situation) changes in a given manner. New situations are reached through actions, external events, or changing perceptions.

4.3 Differentiation of Action Types

We argued in section 3 that an improvement-oriented repository process model must support three different kinds of actions: product process actions, control actions (feedback), and improvement actions (change operators). *Product actions* change product objects and correspond to what Madhavji (1992) calls the "object level". They produce intermediate and final product and can be performed according to special product situations such as "baseline requirements model approved". Action examples of this kind are editing source file, or compiling or mapping requirements into design. *Control actions* may be applied in special process situations, e.g., "intermediate product completed". Examples of such actions are measurements, reviews or audits. *Improvement actions* change the process definition according to results of control actions, experience or general new insights; they therefore correspond to Madhavji´s meta level changes.

The term "action" refers to something that actually happens; in our IRDS extension, we therefore speak of "actions" when we talk about the data layer. In a process definition, we describe the kinds of product, control, and improvement actions by corresponding *product methods*, *control methods*, and *improvement methods.*

While all of these subtypes are basically "just actions", a careful analysis of their interrelationships and of the different situations in which they apply, may substantially increase the level of support offered by the repository. Formally, the three action types are gained by specialization of the class *Methods*, as shown in fig. 9. As part of defining the semantics of each kind of action in the repository, the objects used as input for the specialized actions as well as the output produced by them are specializations of the class *Object*. Similarly, the situations in which particular actions can be applied are defined by specializing the class *Situations* (not shown in fig. 9).

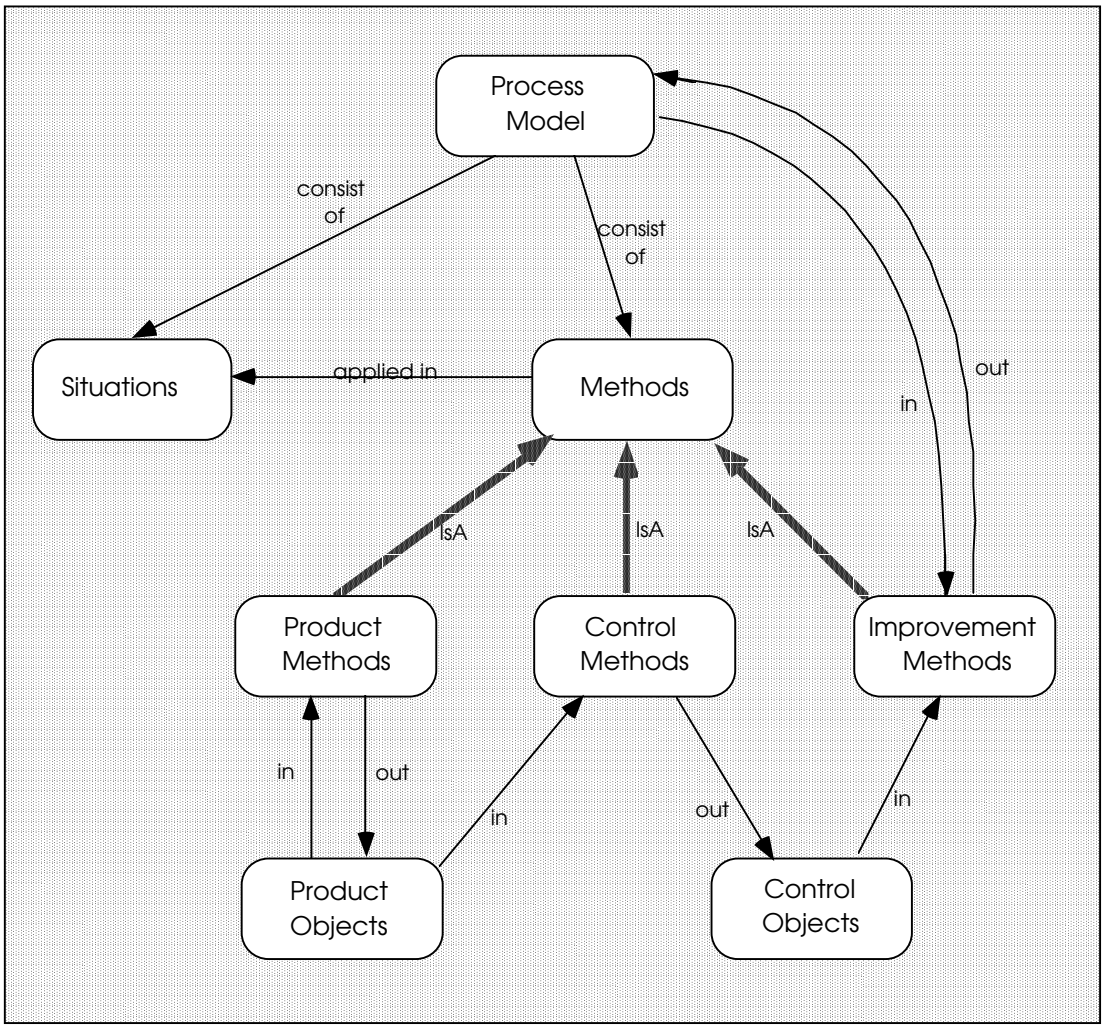

Fig. 9: Three kinds of generic process actions and their input/output objects

4.3.1 Product Actions

Product actions change the states of the product objects. Methods for them are the ones normally considered by process modelling. For example, the software process is often divided in lifecycle phases such as requirements engineering, design in the large, design in the small, implementation and maintenance. Each of them is then modelled at a finer level of granularity. At the lowest level of abstraction, individual possibly tool-supported actions such as discussed in section 2 for the situation "requirements model approved" and the decision class "mapping requirements to designs" are defined. Examples of basic product action types are the two different mapping methods for mapping SA bubbles into a transaction design.

As pointed out earlier, product actions can be defined for different viewpoints and different modeling scopes, according to a genericity hierarchy.

Executing a product action has several effects. First of all, it obviously changes the product. This change, and therefore the new state the product is in after the action is executed, does not only depend on the action used but also on the product states before the

action was applied. Hence, to make the production process traceable, also the dependencies between the different object states must be recorded.

The product produced by a product process action serves as an input for the control actions. However, the usual product process models contain little in the way of facilitating this relationship. Few of them include quality goals (also called nonfunctional requirements) directly in their framework of guiding process execution.

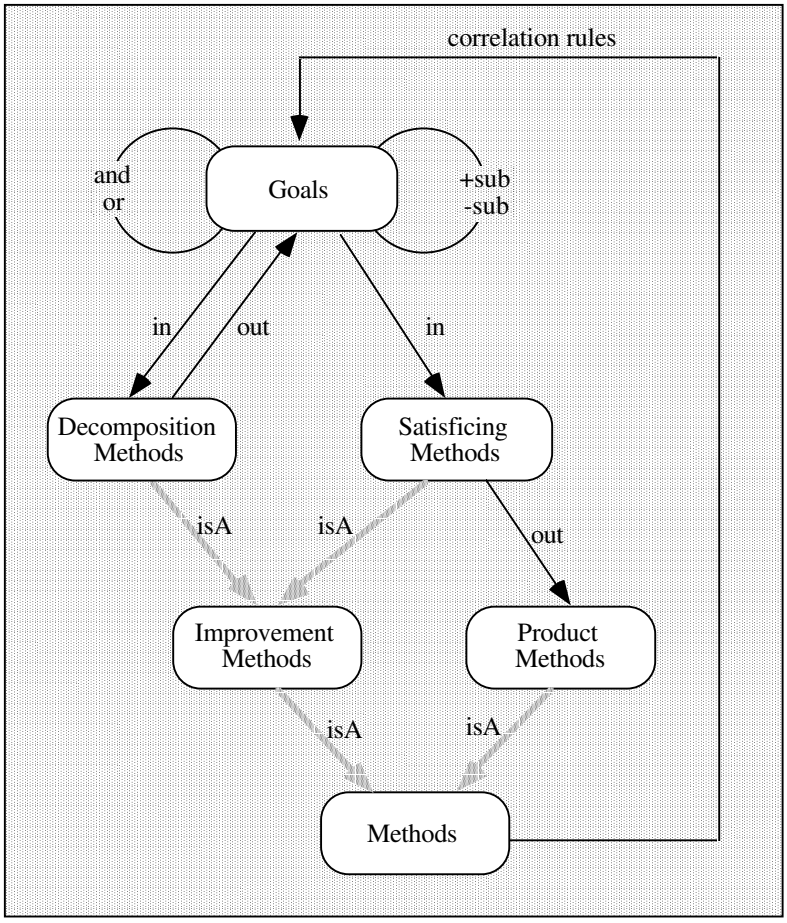

Fig. 10: Metamodel for quality-driven design process.

Figure 10 is our semantic network re-interpretation of a model proposed in a search-tree formalism by Mylopoulos et al. (1992) for representing and using nonfunctional requirements in the design process. It distinguishes three kinds of goals, and three kinds of methods which create these goals as well as links between them:

- Decomposition methods offer one possible scheme of subgoals that have to be achieved to achieve the overall goal.
- Satisficing methods describe options how to achieve these goals.
- Argumentation methods define the relationships between subgoals and methods — e.g., the fact that two methods exclude each other and that there are arguments for both of them, thus requiring a human decision. In figure 10, this part of the model is omitted since we use a different approach to argumentation (cf. section 4.4).

Links can be of type *and/or* (e.g., a conjunction of subgoals must be satisfied to satisfy a goal) or of type +/- (e.g., a method contributes positively or negatively to a goal). Correlation rules state unintended side-effects where the application of some method to achieve one goal also influences other goals.

The approach is qualitative in the sense that only positive and negative influences are recorded; trade-offs in conflict situations have to be made by humans. In contrast, Sylla and Arinze (1991) propose a method for selection of reusable process components under multiple quantitatively expressed quality criteria.

4.3.2 Control Actions

The definition of product action offers the technology to ensure that the production process can be defined (of course, this still has to be actually done!). This definition is the basis for applying control actions which verify or validate intermediate products or the process execution itself. The result of a control action is recorded in a control object. The control object therefore contains information about the execution of the process; for instance, if a desired quality was produced in the first place. This information is essential for process improvement.

There exist different kinds of control actions within the software development process. In early stages such as requirements engineering, reviews, audits or walkthroughs are used. This is based on the fact that automatic control of product quality (the requirements engineering document) is nearly impossible. With growing formalization, some (but never all!) of the control activities can be automated. Many automated control actions during the implementation phase use metrics to obtain indirect software quality information. Which particular metrics is used depends on the quality goals which have to be reached.

The Goals-Question-Metric (GQM) model of TAME (Oivo and Basili 1992), shown in fig. 11, nicely complements fig. 10 by relating metrics to a set of given goals. TAME relates the notions of goal and method to each other through the intermediate construct of questions that *need* to be answered in order to evaluate the contribution of a process execution towards a goal, and that *can* be partially answered by specific metrics. Thus, while the model of fig. 10 describes the *planned* relationships between goals and methods through a goal-based product process model, the one in fig. 11 measures the *actually achieved* relationships between the application of a given method and various goals (including but not only those it was originally designed to satisfy) — thus providing feedback on the quality of process definition.

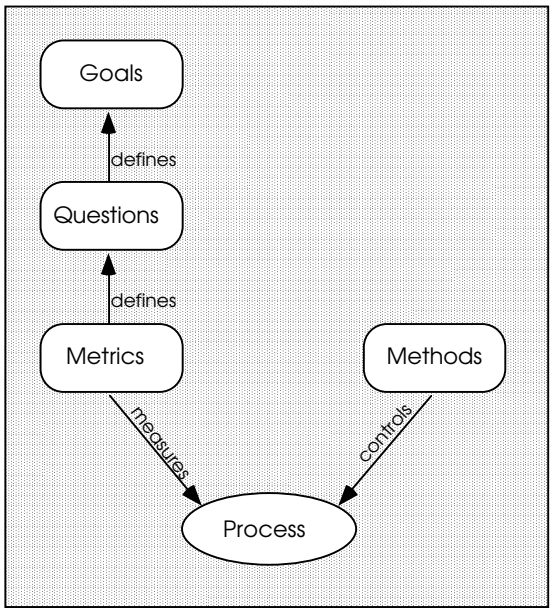

Fig. 11: Process control according to the TAME meta model

Besides these well-known control methods, a second, less discussed kind is as important for process improvement. The activity of recording the process execution forms the basis for this second class of control actions. The execution of the process might not follow the defined process model, due to decisions made by human developers or due to failure to follow the process rules by either humans or tools.

The fact that a process was not executed as defined is an important observation. By analysing the reasons which lead to a process violation new insights for process improvement may be gained. But this is only possible if the process knowledge was recorded in a suitable way. So for this kind of control action, the differences between the defined process and the actually exectuted process must be made visible, together with rationales for these aberrations.

Another use of recorded process knowledge could also lead to process improvement. If, after the product is finished, it is recognized by the user that a certain quality was not achieved, changes to the product must be made. During this change, control actions must make sure that all actions which lead to the missing quality are localized, tracing back the process history to the action where the production of quality failed for the first time. The decision which lead to the execution of this action must then be revised. In retrospect, this makes the actual process execution to a certain degree undefined (we have changed the process definition post-mortem). Again the differences between the (new) defined process and the execution (the changes in this case) could serve as the basis for process improvement. Assuring that maintenance activities (which are product activites) change the process execution and assuring that the violation of the process definition leads to a control object builds the basis for process improvement.

Summarizing both recording the difference between process execution and process definition (in the first place and due to changes afterwards) and the normal control activities applied on products (reviews, metrics, audits etc.) lead to control objects which serve as input for the improvement actions. Combined with the definitional facility of fig. 10, the basic

TAME framework of fig. 11 works in all three cases, provided that the notion of Goals in the figure also includes process goals.

4.3.3 Improvement Actions

Improvement actions are used for changing the process definition. "Improvement" is meant here similar to "change management" (Madhavji 1992), but with the added demand of explicitly considering quality factors in making and recording the process definition change.

In contrast to the product and control actions which deal with product and process objects at the execution level, improvement actions affect the process models defined at the second level of fig. 7. The situation on which an improvement action is based does not only comprise the current process definition and control object information but also existing goals, policies, laws and resources. These are represented as special objects within the process model. Applying an improvement action either leads to an improved process model, or to a decison that according to the facts, no improvement is necessary. Since the decision to make no improvement could affect later decisions, it must also be recorded. An improvement could effect the project, product or organization specific process model.

Like product and control actions, improvement actions can only be applied in special situations. Our model distinguishes three possible kinds of improvement situations (fig. 12).

The first kind are situations where *lack of quality* of an (intermediate) product was recognized by a control action. In this case there exist control objects which characterize the improvement situation.

The second kind covers situations where a *violation of the defined process* was recognized. Not the missing quality of the product is the motivation for process improvement, but the unexpected performance of the process made to gain better quality in the first place.

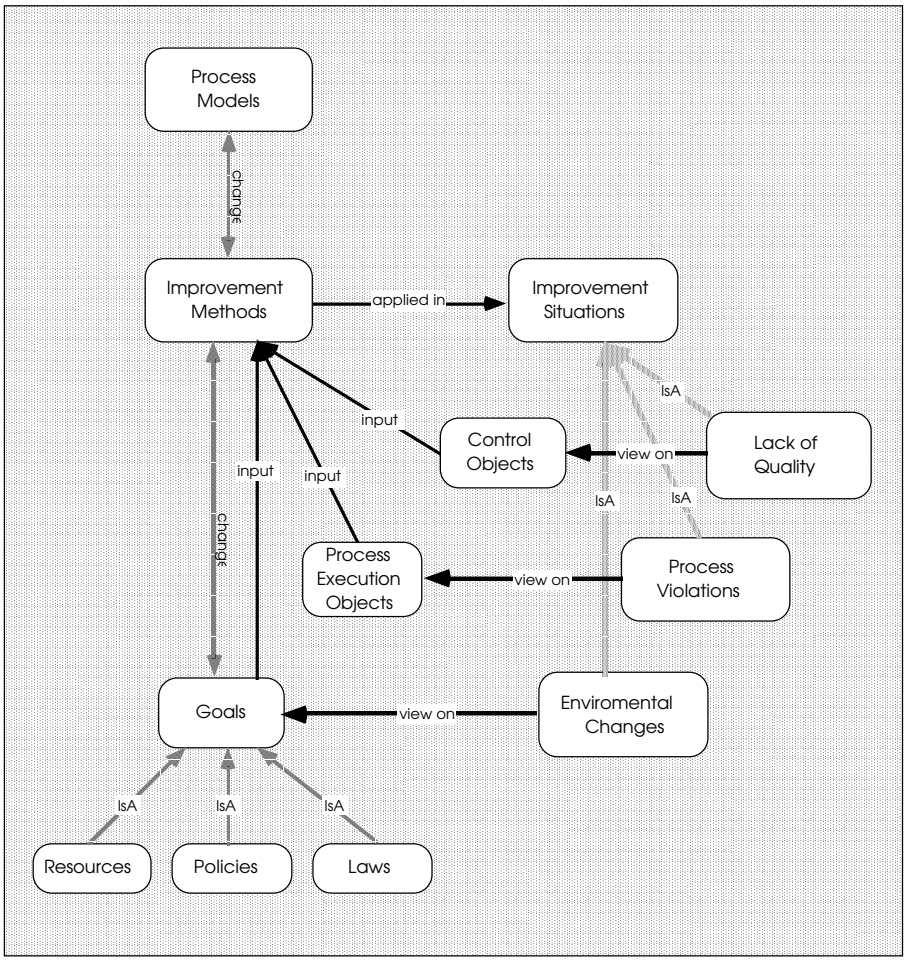

Fig. 12: Improvement Methods with different improvement situations

In contrast to the first and second improvement situation which are recognized by performing the production process, the third kind of improvement situation is raised by *changes in the environment*: changes of goals, policies etc. For instance, changing a policy must lead to improvement actions by which the new policy is integrated into the corresponding process model. Since the goals, policies, laws and resources are represented within the process models, changes made to them (which are modelled as improvement actions, often on a higher level of the genericity hierarchy) constitute improvement situations which assure that the process models are adapted to these changes.

Besides different improvement *situations* there exist different kind of improvement *actions*. Like product and control actions, improvement actions can have a different scope with respect to organizational or product genericity. There are actions that affect the whole organization, but also improvement actions for special product processes or even special project processes exist. As in PRISM (Madhavji and Schäfer 1991, Madhavji 1992), improvement actions defined at different genericity levels deal with the adoption of the process models on this level according to changes that effect this modelling level. This is necessary since there are needs for process improvement which can be satisfied by improving the project model; for instance, if a project goal wasn't reached. Other improvement situations — less quality of a special product caused by errors in a specific process phase —

need an improvement of the product process model, e.g an additional inspecition. The change of an organizational policy is an example in which the organization process model must be improved. So, there exist situations in which changes at different genericity levels must be made. It is up to the process improvement model to guarentee that the improvements are always made at the right level of the genericity (as low as possible, according to the organizational principle of subsidiarity).

The different kinds of situations, the different kinds of improvement actions, and the improvement at different genericity levels make it necessary to differentiate between product process model and improvement process model. Specifically, to *define* the improvement process, it is necessary to investigate suitable metalevel operators that offer structure to process improvement and thus facilitate quality-oriented change control — in essence, providing maturity stages for the improvement process as well as the product process: has improvement been actually accomplished (repeatability)? how does a general improvement strategy look like (defineability)? how to manage change of improvement strategies (manageability)?.

4.3.4 The Integrated Quality-Oriented Action Model

Figure 13 shows how the three different kinds of methods for defining product, control and improvement actions are related to each other through the explicit management of quality goals.

Product processes are defined and executed using goal-oriented meta operators such as goal decomposition and satisficing. Experiences with these process definitions are captured through goal-driven control methods in the spirit of TAME. Improvement is carried out in a structured manner, again under explicit consideration of goals and with the use of structured meta operators; moreover, we recall that, similar to PRISM (Madhavji 1992), this model operates at multiple interacting levels of a genericity hierarchy of products and organizational units. Further semantics is provided to our model through the corresponding specialization of their associated situations, input and output objects, including the goals themselves

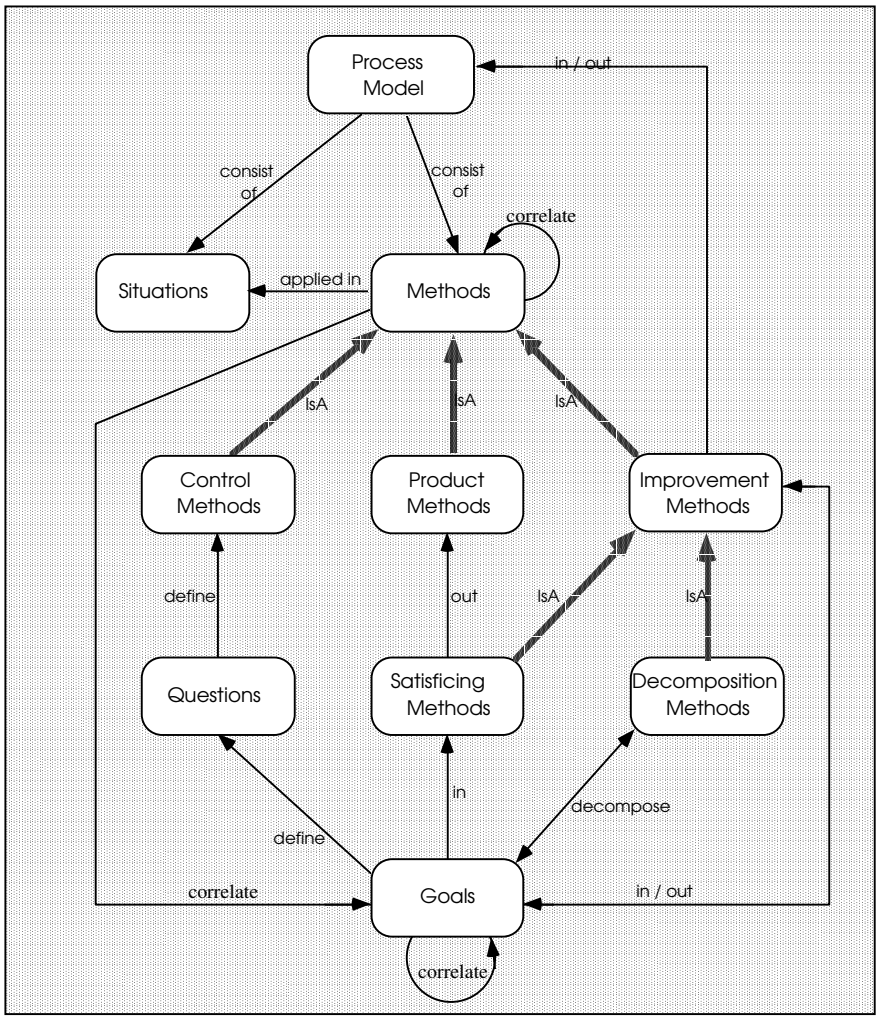

Fig. 13: The integrated meta process model

4.4 Integration of Teamwork

According to the TQM philosophy (cf. Oakland 1990 or Deming 1986), there is no process improvement without teamwork. Each person involved in a project must have the chance to understand the process execution and process evolution. Recording process knowledge could serve as a basis for later understanding of process execution and evolution.

We do not see the process execution as an automatic transition from one state to the next one, but as a cooperation between agents. Taking a speech-act perspective (Winograd and Flores 1986), our model organizes this cooperation in conversation structures which lead to decisions that define and select actions. Decisions select among a set of actions to be performed by an (set of) agents (persons or machines). Of course, there are situations in which the way of acting should not be variable and hence strictly defined. But in general, the way of executing the process should be decided during its execution.

Since conversations may deal with knowledge not contained in the project-specific process definition (e.g., a persons is ill), it is not possible to fully predict a process execution. The process definition proposes a ranking between the actions which could be applied in a given situation. But the decision which of them is actually applied is the responsibility of the agents executing a process. As a result of the conversation, a decision could be made which

chooses another action than the one with the highest a priori ranking. Even a decision for an action which was undefined for the given situation is conceivable.

The performance of an action may require special skills. Thus, decisions depend not only on the current process state but also on the availability of suitable agents which have these special skills. The conversation about the action to be performed in a given situation must therefore consider the skills needed for each action and the availability of suitably skilled agents.

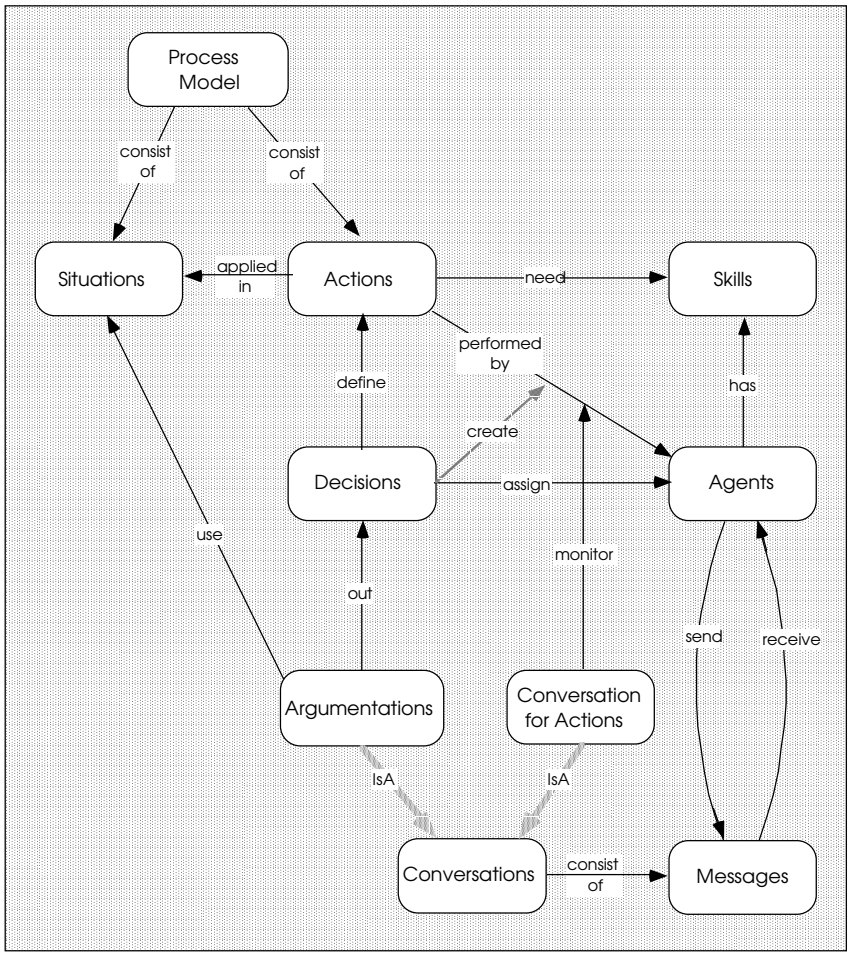

Fig. 14: The meta process model including teamwork aspects

How can we integrate teamwork support at the meta level of the schema description layer? Jarke and Rose (1992) present the CAD^o metamodel as a formal framework in which decision classes, agent teams and their conversational structures can be uniformly represented and related to each other as well as to product and goal objects. Its integration with our meta process model provides the required team-oriented quality information systems model.This extended meta model (cf. fig. 14) serves as a basis for the definition of generic specialized processes (e.g. product and improvement process) and for organization, product and project specific models. The integration of conversations (done by exchanging messages), decisions, agents and skills into the meta process model leads not only to the possiblity to offer suitable process support at the lower modelling level, but leads to a recording of process knowledge which makes it possible to understand process execution also in the future.

Besides the conversation meta-type Argumentations (of which the IBIS protocol is one possible instance see below) which is used for preparing decisions, the model also contains

the meta-type Conversations for Action which is intended to organize the distributed execution of these decisions by assigning and monitoring the application of methods by specific agents. Again, the semantics for these two kinds of conversations is provided through appropriate input and output types and message structures (Rose et al. 1991).

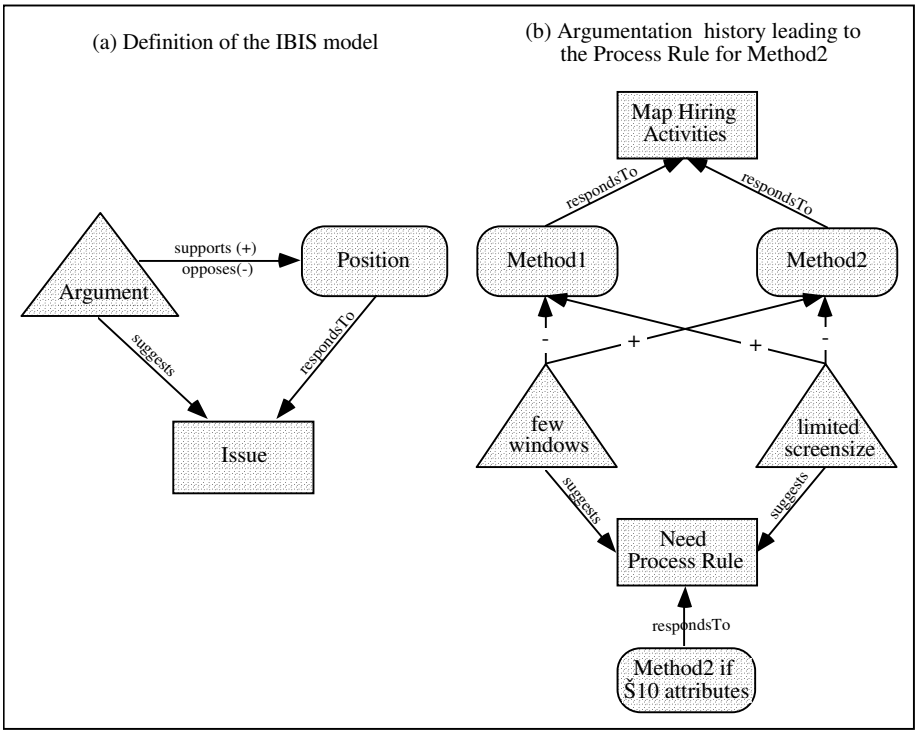

Fig. 15: IBIS model of argumentation: definition and application to process improvement

Structured conversations do not only play a role when executing process definitions but also when deciding on improvement actions. Let us apply this to the example in section 2. Fig. 15(a) shows a schema-level definition of Rittel´s Issue-based Information Systems model which has gained wide visibility through its use for structuring the early phases of design deliberations in the gIBIS hypertext system (Conklin and Begeman 1988) and its more formal derivatives (Ramesh and Dhar 1992). Figure 15(b) shows how a group of developers use IBIS to develop a process definition from a discussion of the two methods used in the example: mapping to two transactions (Method1), or to one transaction (Method2). Arguments for the latter include the need for few windows (see customer complaint), arguments for the former the limited screensize which may force splitting windows. These arguments suggest the issue of defining a rule which can serve as a process definition in such situations. The group decides that in situations where at most ten attributes have to be shown on the screen only Method2 is executable, otherwise only Mehtod1. Doing this the process model was improved.

5 Conclusions

The contribution of this paper has been twofold. It characterizes each stage of the SEI process maturity model in terms of what repository features it must be able to support, and it presents a comprehensive quality-oriented process model for an advanced stage of this model.

The model draws on previous work related to representing and using quality requirements, goal-oriented usage of metrics, and change management, but goes beyond them in two ways:

- On the ontological side, the process model makes a clear distinction among three kinds of actions which, together, comprise an improvement-oriented environment: product process, control, and improvement. As a basis for their correct usage in a repository, these action types are given a formal semantics through corresponding specializations of their enabling situations, input and output objects. This semantics is further differentiated according to subclasses for each kind of action. This, plus the common explication of quality goals in all action types, also provides a deep integration of the process model, and contributes towards the goal of TQM to increase quality awareness throughout the process cycle.
- On the epistemological side, we carefully distinguish the abstraction principles of classification (meta layers of linguistic type systems) and genericity (organization- or product-oriented scoping of model validity). This brings us closer to current repository standards such as IRDS, thus providing a migration path. It also allows object-oriented principles such as inheritance to be exploited in technical support of process model hierarchies, thus promoting conciseness and consistency.

While these considerations provide a certain theoretical validation of our approach, several ongoing activities are geared towards a practical validation.

In the software engineering domain, a major chemical company is using the model to define and measure quality of their knowledge-based process control software that supervises the production of detergents. The main goal here is to refine control methods, both at the object level of the actual production process and for the expert systems software that controls this process. The model has proven rather valuable to disentangle this conceptually confusing situation. A second usage is made by a CASE vendor who employs the model to build re-engineering tools for relational database software, with quality goals such as conceptual clarity, performance, and minimal cost of database re-organization in mind. The main goal here is to correlate the activities of the reverse and forward engineering tools of the environment.

Outside the software engineering area, a consortium of industrial engineers has adopted the model, with the goal of unifying terminology and providing repository support for the data and process models underlying the different kinds of quality measures adopted throughout the product life cycle, ranging from quality functon deployment in design through statistical process control during production to service report analysis during maintenance. Judging from the limited experiments completed so far (modeling a vacuum cleaner life cycle), the model seems to be the only one among several candidates considered that at least captures all the domain-independent aspects.

Based on these ongoing experiences, several aspects of the proposed framework are further elaborated, especially concerning the human-machine division of labor and the requirements they place on repository organization and interface. From the representational viewpoint, monotonic genericity may not be enough for intuitive understanding in large multi-criteria process models; organized over-simplification gives the user a broad-brush picture first and only on demand explores the maybe contradictory details. From the interface viewpoint, the two-way interaction of formal and informal representations remains a major problem, especially in the team setting. Finally, we were only able to present a few specific operators for process improvement; a comprehensive meta-process theory is not in sight. It will probably have to be based on a much improved understanding of many specific domain models and their underlying principles. Together with partners in England, France, Greece, and Sweden, we are currently embarking on an ESPRIT project called NATURE to address these issues in depth.

References

Aslett, M. J.; Mellgren, D.; Yan, Y. F.; and Pietri, F. "ASPIS: A Knowledge-Based Approach To Systems Development." In Proceedings of the ESPRIT Conference, Brussels, Belgium, 1989, pp. 334-344.

Basili, V. R.; and Rombach, H. D. "The TAME Project: Towards Improvement-Oriented Software Environments." *IEEE Transactions on Software Engineering*, Volume 14, No. 6, 1988, pp. 758-773.

Blaine, L.; Goldberg, A.; Pressburger, T.; Qian, X.; Roberts, T.; and Westfold S. "Progress on the KBSA Performance Estimation Assistant" In *Proceedings 3rd Annual Knowledge-Based Software Assistant Conference,* August 2-4, 1988, pp. 100-116.

Chen, P. P. S. "The Entity-Relationship Approach: Towards a Unified View of Data." *ACM Transactions on Database Systems,* Volume 1, No. 1, 1976.

Conklin, J.; and Begeman, M. L. "gIBIS: A Hypertext Tool for Exploratory Policy Discussion" *ACM Transactions on Office Information Systems* Volume 6, No. 4, 1988, pp. 303 -331.

DeMarco, T. *Structured Analysis and System Specification* Prentice Hall, Inc. Englewood Cliffs, 1978

Deming, W. E. *Out of the Crisis*. Massachusetts Institute of Technology, Center for Advanced Engineering Study, Cambridge, Massachusetts, 1986

Dowson, M. "Integrated project support with IStar" *IEEE Software,* Volume 4, No. 4, 1987, pp. 6-15.

Green, C.; Luckham, D; Balzer, R.; Cheatham, T.; and Rich, C. "Report on a Knowledge-Based Software Assistant". Reprinted in C. Rich and R. C. Waters (eds) *Readings in Artificial Intelligence and Software Engineering,* San Mateo, Ca: Morgan Kaufman, 1986, pp. 377-428.

Grosz, G; Rolland, C. “Why and How should we hide Conceptual Models“ *Proceedings 3rd Int. Conf. on Software Engineering and Knowledge Engineering (SEKE), Skokie, USA, 1991, pp. 28-33.*

Hahn, U.; Jarke, M.; and Rose, T. "Teamwork Support in a Knowledge-Based Information Systems Environment". *IEEE Transactions on Software Engineering,* Volume 17, No. 5, 1991, pp. 467-482.

Hauser J. R.; and Clausing, D. "The House of Quality" *Harvard Business Review,* May-June, 1988, pp. 63-73.

Humphrey, W. S. *Managing the Software Process*. Addison-Wesley, 1990.

IEEE *Software Engineering Standards.* Institute of Electrical and Electronics Engineers, Inc., 1987.

ISPW *Proceeedings of the 6th Int. Software Process Workshop.* Hokkaido, Japan, October, 1990.

IRDS *Integrated Resource Dictionary Standard*, 1988, ANSI X3.138 und ISO WG3

Jarke, M. “DAIDA: Conceptual Modeling and Knowledge-based Support of Information Systems Development Processes“ In *Technique et Science Informatiques*, Volume 9, No. 2, 1990, pp. 122-133.

Jarke, M. (ed.) *ConceptBase V. 3.1 User manual*, Technical Report, RWTH-Aachen, Germany, 1992.

Jarke, M; Jeusfeld, M.A.; and Rose, T. "A Software Process Model for Knowledge Engineering in Information Systems" *Information Systems,* Volume 15, No. 1, 1990, pp. 86-115.

Jarke, M; and Pohl, K. “Information Systems Quality and Quality Information Systems“ In *Proceedings of the IFIP WG 8.2 Working Conference on The Impact of Computer Supported Technologies on Information Systems Development*, Minneapolis, Minnesota, USA, 14-17 June, 1992, pp. 345 -375.

Jarke, M.; and Rose, T. "Specification Management with CAD^{O}". In Loucopoulos, P., Zicari, R. (eds.): *Conceptual Modeling, Databases, and CASE,* Chichester, UK: John Wiley, 1992.

Johnson, A.M.; and Malek, M. "Survey of Software Tools for Evaluating Reliability, Availability, and Serviceability". *ACM Computing Surveys,* Volume 20, No. 4, 1988, pp. 228-269.

Kaiser, G. E.; Feiler, P. H.; Popovich, S. S. “Intelligent assistance for software development and maintenance“ In *IEEE Software*, May, 1988, pp. 40-49.

Madhavji, N. H. “The Process Cycle“. In *IEE Software Engineering Journal*, Volume 6, No. 5, 1991, pp. 234-242.

Madhavji, N. H. “Environment Evolution: The Prism Model of Changes“, *IEEE Transactions on Software Engineering*, Volume 18, No. 5, May, 1992, pp. 380-392.

Madhavji, N. H.; and Schäfer, W “Prism = methodology + process oriented environment“ *IEEE Transaction on Software Engineering*, Volume 17, No. 11, November, 1991.

Madhavji, N. H.; Gruhn, V.; Deiters, W.; and Schäfer, W. "Prism: Methodology and Process-oriented Environment" In *Proceedings 12th International Conference on Software Engineering,* Nice, France, 1990, pp. 277-288.

Miyoshi, T; Togashi, Y; and Azuma, M. "Evaluating Software Development Environment Quality" In *Proceedings 13th International Computer Software & Applications Conference,* Orlando, Fl, 1989, pp. 501-508.

Mylopoulos, J.; Borgida, A.; Jarke, M.; Koubarakis, M.; “Telos: Representing Knowledge About Information Systems" In *Transactions on Information Systems*, Volume 8, No. 4, 1990, pp. 325-362.

Mylopoulos, J.; Chung, L.; and Nixon, B. "Representing and Using Non-Functional Requirements: A Process-Oriented Approach" *IEEE Transactions on Software Engineering,* Volume 18, No. 6, 1992, pp. 483-497.

Oakland, J.S. "Total Quality Management" In *Proceedings 2nd Inernational Conference on Total Quality Management,* Oxford: Cotswold Press Ltd, 1989, pp. 3-17.

Oivo, M; and Basili, V. R. "Representing Software Engineering Models: The TAME Goal Oriented Approach" *IEEE Transactions on Software Engineering,* Volume 18, No. 10, 1992.

Osterweil, L. "Software Processes are Software too" In *Procceedings 9th International Conference on Software Engineering*, Monterey, Ca, 1987, pp. 2-13.

Perry, D. E. “Policy and product directed process instantiation“ In: *Proceedings of the 6th Int. Software Process Workshop*, Hokkaido, Japan, October, 1990.

Perry, D. E.; and Kaiser, G. E. “Models of software development environments“, *IEEE Transaction on Software Engineering*, Volume 17, No. 3, 1991, pp. 283-295.

Potts, C; and Bruns, G. "Recording the Reasons for Design Decisions" In *Proceedings 10th International Conference on Software Engineering,* Singapore, 1988, pp. 418-427.

Ramesh, B.; and Dhar, V. "Process-Knowledge Based Group Support in Requirements Engineering", *IEEE Transactions on Software Engineering,* Volume 18, No. 6, 1992, pp. 498-510.

Rose, T.; Jarke, M.; Gocek, M.; Maltzahn, C.G.; and Nissen, H.W. "A Decision-based Configuration Process Environment", Special Issue on Software Process Support, *IEE Software Engineering Journal,* Volume 6, No. 5, 1991, pp. 332-346.

Sagawa, J.W. "Repository Manager Technology", *IBM Systems Journal,* Volume 29, No. 2, 1990, pp. 209-227.

Selby, R. W.; Basili, V. R.; and Baker, T. F. "Cleanroom Software Development: An Empirical Evalutation." *IEEE Transaction on Software Engineering*, Volume 13, No, 9, 1987, pp. 1027-1037.

Sirvent, O.; and Dupont, C. "SAMSON: An Expert System Devoted to the Help of Software Assessment". In *Proceedings ESPRIT ´90 Conference,* Brussels, Belgium, 1990, pp. 264-273.

Smolander, K.; Lyytinen, K.; Thavanainen, V.-P., Martiin, P. “MetaEdit - A flexible graphical environment for methodology modelling In Andersen, R; Bubenko, J; Solvberg, A; (eds) *Advanced Information System Engineering*, Springer-Verlag 1991, pp. 168-193.

Sylla, C.; and Arinze, B. "A Method for Quality Precoordination in a Quality Assurance Information System". *IEEE Transactions on Engineering Management,* Volume 16, No. 3, 1991, pp. 245-256.

Thomas, I. "Writing Tools for PCE and PACT". In *Proceedings ESPRIT ´88 Conference*, Brussels, Belgium 1988, pp. 453-459.

Winograd, T.; Flores F. “Understanding Computers and Cognition: A New Foundation for Design“ Ablex, Norwood, NJ, 1986.