Information Systems Quality and Quality Information Systems

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

Abstract. *The quality of information systems (IS) has not been a major impact of computer-aided software engineering so far. We characterize the requirements for quality-oriented CASE in information systems and present a stepwise procedure how something like Total Quality Management can be achieved in software environments through process-oriented repository technology.*

Keyword Codes: D.2.2; H.4.0; K.6.0
Keywords: Software Engineering, Tools and Techniques; Information Systems Application, General; Management of Computing and Information Systems, General

---

[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.

1 Introduction

Total Quality Management (TQM) is clearly one of the buzzwords of the day. The information systems (IS) market has not been spared the excitement spurned by concepts popularized in Japan in general industry. For example, in its ESPRIT III workprogram, the European Community uses aspects such as quality of the information or quality of the interface as an organizing principle. However, a closer look shows more confusion than clarity what is really meant by quality and how to achieve and maintain it. The main emphasis is still on new product technologies such as "intelligent" databases or "cooperative" user interfaces. These techniques, however, are only small facets of a much larger phenomenon.

In the area of computer-aided software engineering (CASE), quality management and improvement-orientation have not been the central concern so far. CASE tools partially automate tedious tasks but are currently ill-suited for the requirements of long-lived information systems with continuously increasing user demands on their functionality. Studies with well-known CASE packages show an emphasis on productivity and enforcement of rigid standards, rather than on support for team-oriented continuous improvement (Norman and Nunamaker 1989). Our recent survey of 37 German CASE user companies (Zens 1990) showed that the major *expected* impact of CASE was improved maintainability but that *satisfaction* of this expectation is limited by CASE coverage of life-cycle phases and by the degree to which the introduction of CASE improves methodological consistency and communication within software teams. Good experiences with automation and local method standardization does not account for much in a situation where upwards of 80% of software costs go into maintenance. Few positive contributions were found for quality factors such as security, product standardization, reusability, or portability, and schedule and cost control.

Work on software quality contol and assurance has mostly focused on software as a 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 CASE environments.

This paper investigates the requirements induced by the philosophy of Total Quality Management for Information Systems Software, both for CASE tools in general and for repository technology in particular. Technological requirements are derived from organizational goals and methods of TQM but we do not study these organizational aspects directly.

The paper is organized as follows. Section 2 attempts to establish the relationships between work on quality manegement in general, software quality control and assurance, and the special case of information systems software. Section 3 employs a simple information systems development example to illustrate shortcomings of current CASE environments with respect to quality improvement aspects. In section 4, a stepwise approach for removing these shortcomings is presented which gradually extends the current repository concept to that of a quality information system. Section 5 summarizes our contributions and draws some

conclusions concerning the future relationships between CASE and information systems quality.

2 Information Systems Quality

Like all rapidly evolving areas, the quality management domain is full of ill-defined concepts. We follow the IEEE Standard (IEEE 1987) which provides the following definitions for *software quality*:

1. The totality of features and characteristics of a software product that bear on its ability to satisfy given needs (definiton of quality)
2. The degree to which software possesses a desired combination of attributes
3. The degree to which a customer or user perceives that software meets his or her composite expections
4. The composite characteristics of software that determine the degree to which the software is use will meet the expectations of the customer

Many lists of quality attributes for the software product have been proposed (Hausen, Müllerburg and Schmidt 1988; Gilb 1987; Bowen, Wigle and Tsai 1985; Boehm et al. 1978). The probably best known stems from the investigation of a methodology for specifying and evaluating software quality attributes from the Rome Air Development Center and is shown in table 1.

Acquisition Concern	User Concern	Quality Attribute
Performance - How Well Does It Function ?	How well does it utilize a resource? How secure is it? What confidence can be placed in what it does? How well will it perform under adverse conditions? How easy is it to use?	Efficiency Integrity Reliability Survivability Usability
Design - How Valid Is the Design ?	How well does it conform to the Requirements? How easy is it to repair? How easy is it to verify its performance?	Correctness Maintainability Verifiability
Adaptation - How Adaptable Is It ?	How easy is it to expand or upgrade its capability or performance? How easy is it to change? How easy is it to interface with another system? How easy is it to transport? How easy is it to convert for use in another application?	Expandability Flexibility Interoperability Portability Reuseability

Table 1: The RADC software quality consumer oriented attributes
(Keller, Kahn and Panara 1990, p. 146)

Such lists can be complemented by factors specific to particular types of software or application domains. The quality factors are then made evaluable by a procedure similar to

cost-effectiveness analysis in economics or to Quality Function Deployment (Hauser and Clausing 1988) in industrial engineering. According to Keller, Kahn and Panara (1990), consumer-oriented quality factors are first mapped to technical quality criteria. These are then re-interpreted as aggregate *metrics* which can be further decomposed to measurable elements. Numerous activities concerning development of such metrics have been reported. Quality attributes are not independent of each other but can be correlated positively or negatively. Design decisions based on quality therefore require multi-criteria trade-offs.

In this section, we are less concerned with specific metrics. We trace the evolution of quality management in the software area and then try to identify what is special about information systems quality and its management. Our view of this history is summarized in figure 1. It shows clearly the trend from product to process, but also that there have been different attempts to make quality-oriented process management operational which have yet to be integrated. It ends with the question addressed in this paper: what can CASE contribute to all of this?

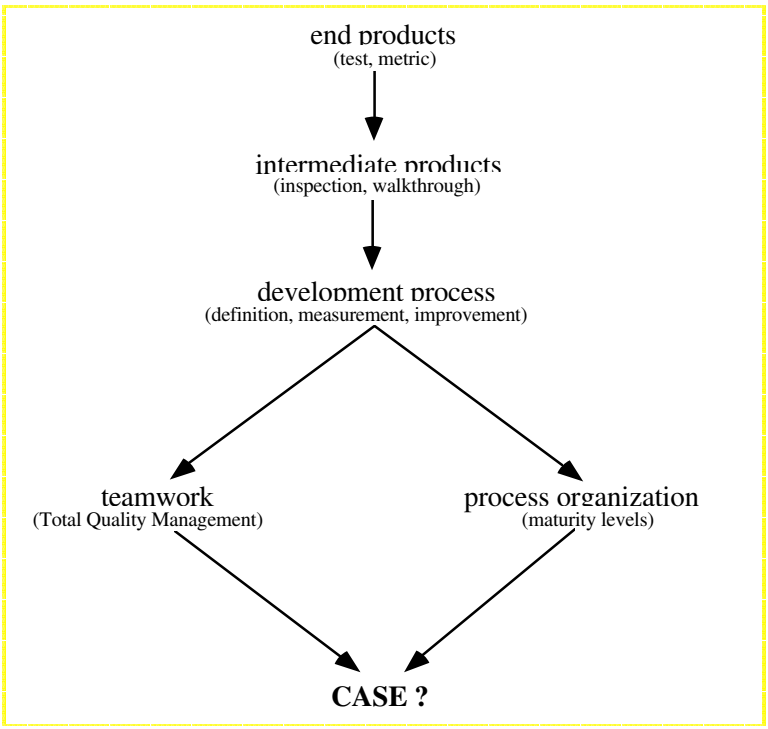

Fig. 1: Evolution of quality management in software environments

2.1 Software Quality Assurance and Software Quality Control

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 asthe operational techniques and activities that are used to satisfy the quality.

Quality Assurance covers the determination of a quality policy and the checking that predetermined Quality Control activities are being properly taken. It defines the framework within which the software development team defines and carries out Quality Control activites. Quality Control can be viewed as the action of checking that a given product (either the final or a intermediate) satisfies a predetermined value or amount.

Testing software products was one of the first quality control activities in software engineering. Until today, it is no question that a software product has to be tested before delivering it. Myers (1976) provides some useful testing definitions:

Testing	the process of executing a program with the intention of finding errors
Verification	an attempt to find errors by executing a program in a test or simulated environment
Validation	an attempt to find errors by executing a program in a real environment
Debugging	diagnosing the precise nature of a known error and then correcting it.

There are two basic tests methods. *White box* tests examine the internal design of the program and therefore require that the tester has detailed knowledge of its structure. *Black box* tests examine the program to find out if it meets the functional requirements. The gap between the two methods is bridged by integration tests, which assemble the parts of the system into a working whole. Long ago, Dijkstra pointed out that "program testing can be used to show the presence of bugs, but never their absence!" (Dijkstra 1970). Therefore, testing could never guarantee the quality of a software product. Test experience showed, that almost always more tests on large systems will find more bugs. It was further recognized that fixing a bug made early in the development cycle later on is very expensive. Therefore, testing a software product can not be the only Quality Control activity. Methods which operate on results of earlier development steps are needed to prevent the propagation of early errors throughout the process.

Formal specifications are one important solution attempt to this problem since they allow several important properties to be analyzed by the means of mathematics and sometimes by computerized tools. However, the traditional informal or semi-formal methods used in practice today allow only for a limited application of these techniques. Structured Walkthroughs (Yourdon 1985) represent a review process in which a designer or programmer leads one or more other members of the development team through a segment of analysis, desgin, code, test that he or she has be produced while the other membes ask questions and make comments about technique, style, possible errors, violation of development standards and other problems. They could be viewed as a first step into process oriented Quality Control.

Inspections were introduced for the dual purpose of improving software quality and increasing programmer productivity (Fagan 1976). They are defined by IEEE standard Std 729-1983 (IEEE 1987) as follows:

1. A formal evaluation technique in which software requirements, design or code are examined in detail by a person or group other than the author to detect faults, violations of development standards, and other problems.
2. A phase of quality control that by means of examination, observation or measurement determines the conformance of materials, supplies, components, parts, appurtenances, systems, processes or structures to predetermined quality requirements.

The basic idea of inspection is to define the output of any intermediate task and to build a group of people (not involved in producing the output) to inspect it. This forces the need of clarifying the structure of the development process. Experience has shown that applying inspection to the development process results in better product quality through less errors (Bush 1990, Fenton 1984). As Fagan (1986) points out, inspection can not only be used to detect defects but also for defect prevention.

IBM´s *cleanroom* software development goes one step further towards introducing team orientation and formality into the development process. Programmer are no longer allowed at all to compile and test their software on a computer. Separate testers interact with developers on the basis of abstract descriptions of the software in a sanction-free environment where the main goal is that the team as a whole achieves good software. This approach discourages incomprehensible programming tricks and should thus improve maintainability. Empirical evidence confirms positive effects (Selby, Basili, and Baker 1987).

2.2 Process Management and Process Maturity

At a more abstract level, several researchers have 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, few if any 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 their 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 2) have meanwhile been widely adopted as an evaluation standard for software suppliers.

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

2.3 Total Quality Management

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[2] . 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.

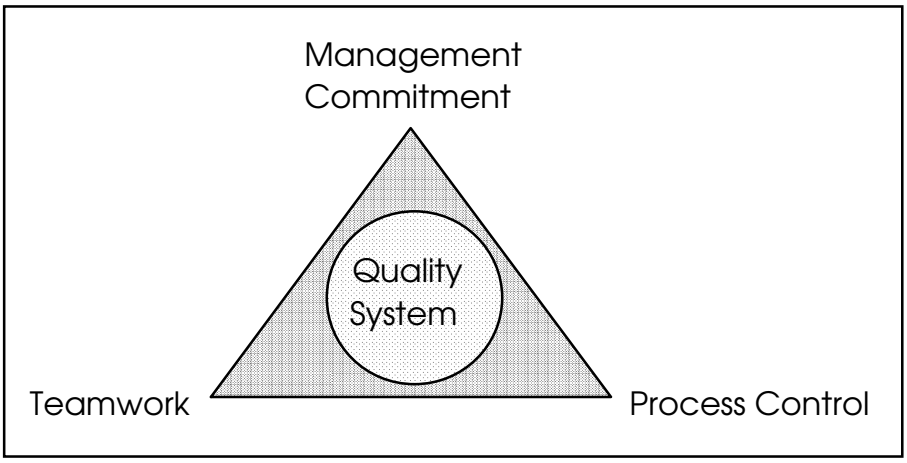

Fig. 2: The TQM triangle (adapted from Oakland 1989, p. 15).

[2] Note the similarity of this idea to the long-term quest for user involvement in software development; the important advantage of QFD is that it tells you how to do it!

Figure 2 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.

If we compare this approach with the process maturity model of Humphrey, 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 Humphrey´s approach, and the teamwork orientation of TQM. Achieving this goal poses a host of organizational challenges but also some important technical questions. In particular, the question arises what CASE can contribute towards these aims. The answer is not an easy one.

In terms of Humphrey´s model, quality aspects are most strongly emphasized in the "defined" level and its transition to the "managed" level. CASE does not contribute much to these improvement steps. Since CASE tools just "work" but have little formal representation of their own capabilities and actions, their precise role in the development process is difficult to describe. There is no representation of the process, let alone of the rationales and goals behind it. Therefore, it is also hard to describe the "desired" state of the process, let alone the transition from current to desired processes. Since CASE tools are mostly hardcoded, it also becomes difficult to migrate them to a new process structure.

In terms of TQM, computer supported cooperative work (cf., e.g., Greif 1988) is the obvious area to look for solutions. However, as we shall see, a more precise analysis is needed what CSCW can actually contribute to the problem at hand. Before we embark on that analysis, however, we make our task a bit easier by concentrating on one particular domain of software engineering, information systems (IS) development. Concentration on such a specific software engineering domain allows for a more precise analysis of the product and the process, also for the definition of domain-specific quality factors. We have chosen information systems because it is an important area and we can build on substantial experience gained in the DAIDA project as summarized in (Jarke et al. 1992).

2.4 The Case of Information Systems

What is special about information systems? Firstly, their constant evolution.We take the perhaps unorthodox view that *an information system is a continuously changing process* of producing information, rather than a product of which some "ideal" finished version could ever exist. Quality factors for IS are therefore determined by the relationship of this process to its surrounding world of other processes.

Second, as shown in figure 3, three *specific worlds of related processes* can be identified for IS: the subject world, the usage world, and the development world. Each may require specific quality attributes and quality control techniques.

In particular, the first of these subworlds is characteristic for information systems. The information system *represents* information about processes in its subject world; relevant quality aspects include accuracy, timeliness, and formal consistency and completeness. Part of these criteria, the semantic integrity constraints, can be automatically supervised by specialized DBMS software. But, as shown by Motro (1989), integrity checking can only cover a limited subset of these aspects by stating that certain data should under all circumstances (under no circumstances) in the database

The IS is also part of its customers´ work processes in the usage world, with *interface*-related goals such as correct functionality, responsiveness, user-friendliness, or security. From a process perspective, the degree of organizational change introduced by an IS accounts for a large part of its acceptance, i.e., perceived quality (Sasso 1991).

Last not least, the system is part of a long *development* history; it is one of many versions in one of many configurations, developed by a changing group of people with evolving CASE tools; goals here include maintainability, portability, configurability, reusability, development costs, and the like.

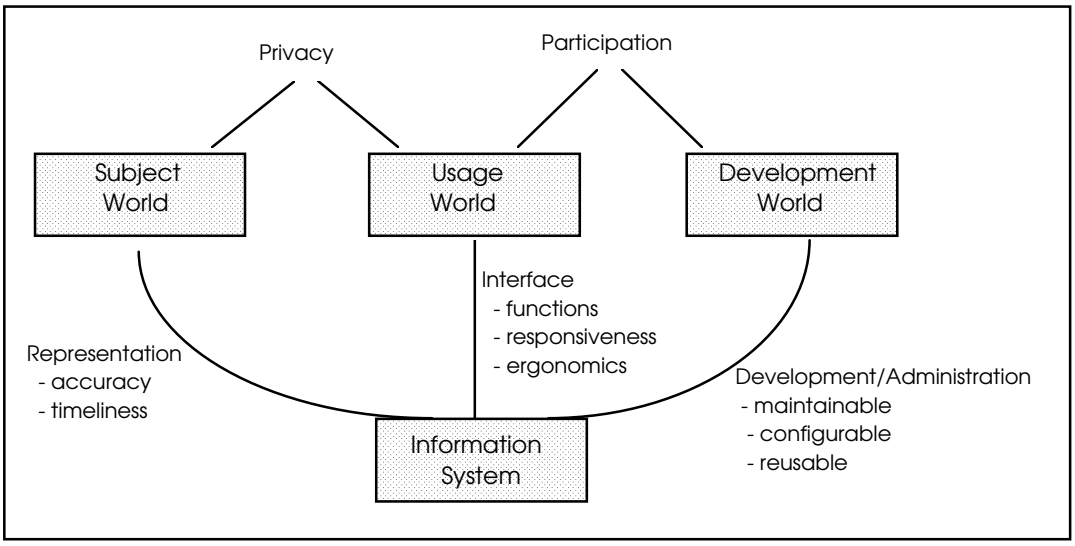

Fig. 3: Organizational principle for IS quality factors.

There are also IS-related quality factors directly between the subworlds which have to be taken into account in describing overall IS and IS development process quality. Privacy of information is a concern that addresses the relationship between usage world and subject world when the subject world contains people or organizations. For example, companies will not contribute data to a pollution control system unless they are sure this information is used only in anonymized form. Similarly, degree of user participation in designs and similar process quality factors associate directly usage and development worlds.

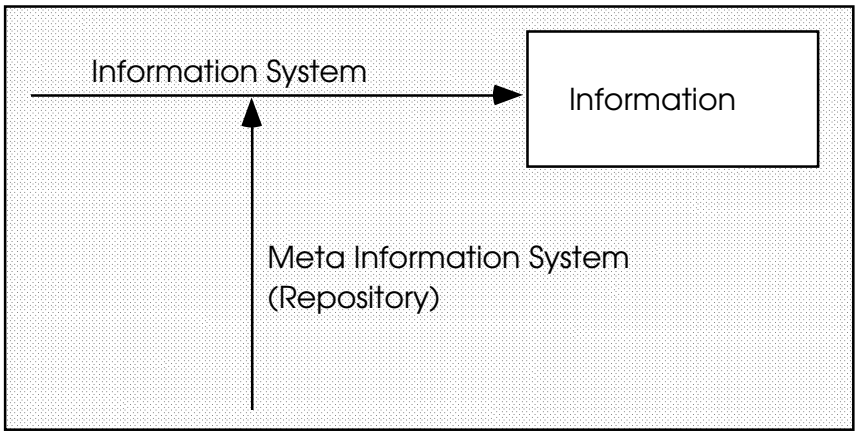

Fig. 4: Information system as a process, repository as a meta process.

Viewing an information system as a process also explains why *repository technology* may be an adequate vehicle to address quality concerns.

Like an information system is a process that satisfies evolving needs for information, a repository or software information system can be seen as a process that evolves information systems (figure 4). Like the information system can only fulfill is role when it rapidly and precisely adapts to the continuously changing demand for its product, the repository can only fulfill the role of quality-orientation in CASE environments if process and improvement orientation are actually supported by the repository technology and if the CASE tools it interacts with can actually exploit this quality orientation.

3. CASE and Information Systems Quality: A (Counter-)Example

Why do traditional CASE tools contribute so relatively little to IS quality? Consider the following example.

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 5 — 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.

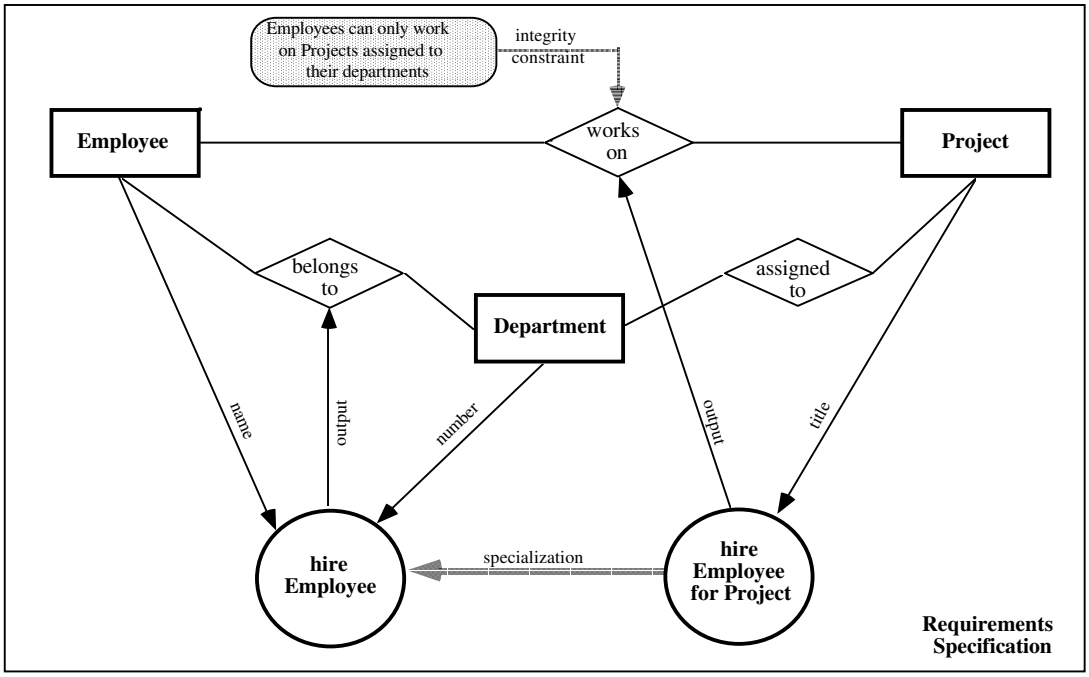

Fig. 5: Requirements specification of the example

Suppose there is a CASE tool which can convert each bubble in such an SA diagram automatically into a transaction specification on a database. This transaction specification is then (say, also automatically) 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 figure 6. Other tools create a (say) relational database from the ER part of figure 5; the system works and is delivered to the customer.

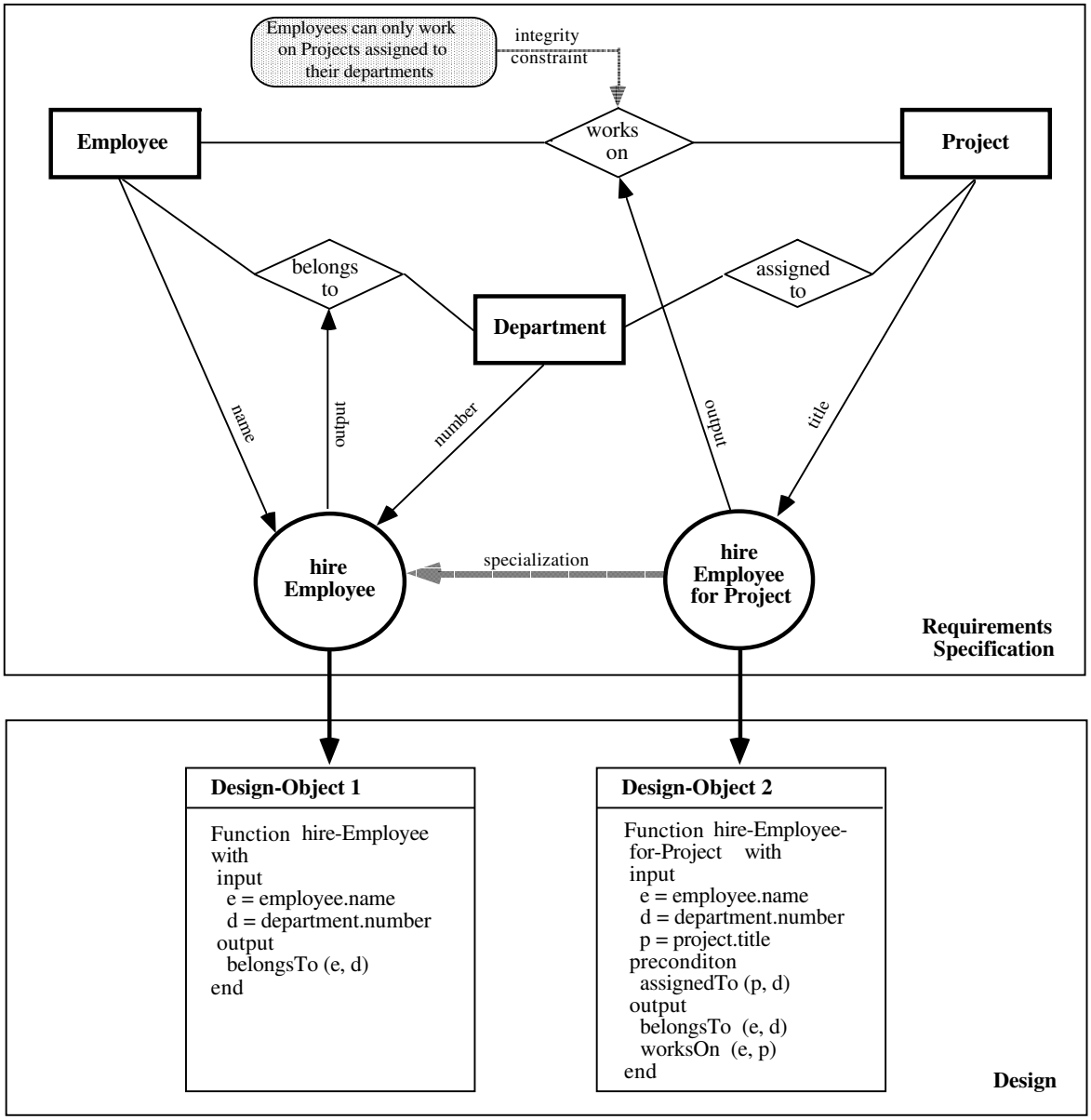

Fig. 6: Design generated from the requirements by the CASE tool.

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 accordingly and, after debugging the result, re-delivers the code to the customer who is happy with this solution (at least for the moment). The result is shown in figure 7.

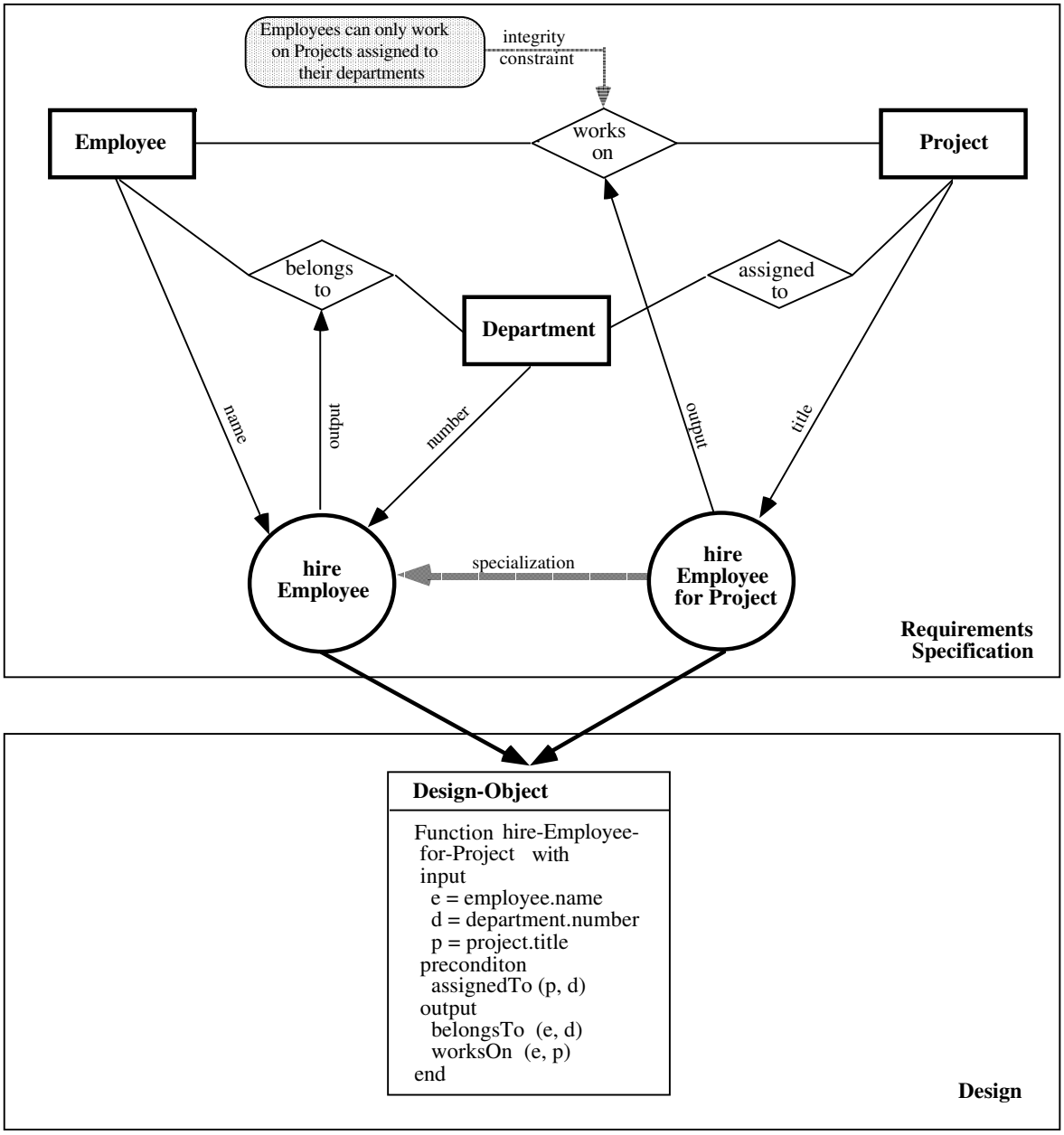

Fig. 7: The mapping solution after the change, as understood by the designer.

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, the answer is: *just about everything!*

(1) The manual change of CASE 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 CASE 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 contiuous 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 CASE tools. However, it does not indicate a single aspect where 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 perhaps overstated but certainly not unrealistic 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 software 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.

4 From Repositories to Quality Information Systems

A repository is the central software information systems around which CASE environments are layered. Standardization efforts such as the European PCTE initiative (Thomas 1988) and commercial announcements such as 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.

This section investigates the usage of repository technology for quality management, stressing the goal of process improvement. We do this in several steps, beginning with fairly well-known ones and ending with rather ambitious ones. We have organized the steps such that they roughly correspond to Humphrey´s process maturity levels but we emphasize teamwork support rather than central management aspects. Each step is illustrated with modifications of the example presented in section 3. In pursuing the steps, we also gradually build up a meta model of quality-oriented information systems development which could be used as a data model for such a repository and has been prototyped in part on our ConceptBase software information system (Jarke 1991).

4.1 Objects and Dependencies: Tracing Development Histories

The first problem mentioned in section 3 was that changes made with different tools must be represented in a uniform process representation as a basis for making the development process traceable and thus possibly evaluable. Starting from the pioneering work of Stallman and Sussman (1977), most repository models allow the documentation of dependencies among design objects as well as of the "author" (human and/or CASE tool). In our case, such a model would document two operations, the mapping from requirements to designs by the CASE tool and the revision of the design by the human designer. The notation chosen in figure 8 for this model is similar to SADT´s idea of representing inputs, outputs (boxes), and controls (rounded boxes) for each process (oval)

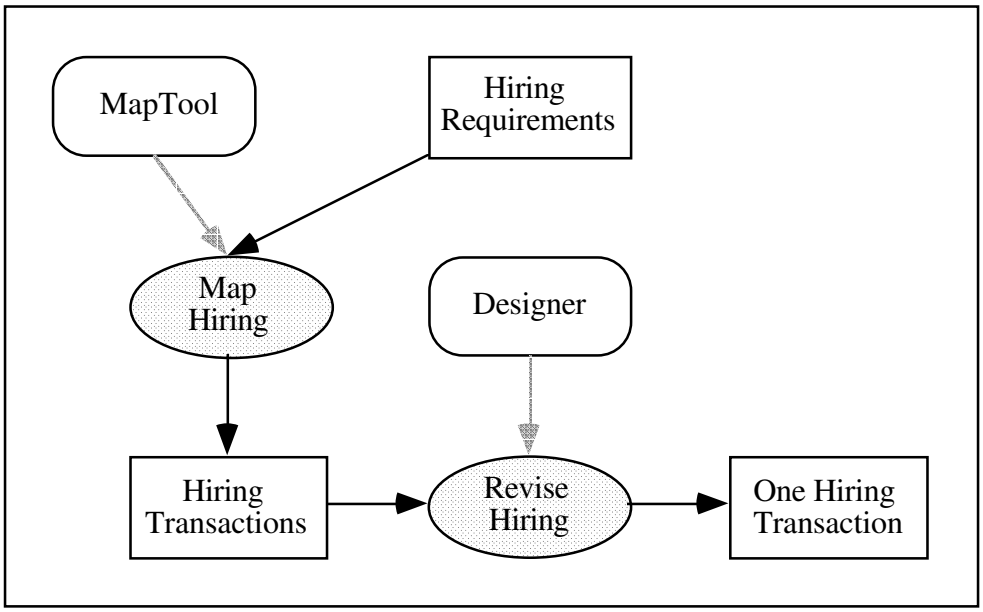

Fig. 8: Documentation of example development process.

The problem remains that even with this model, we have lost the *direct* relationship between the requirements model and the final design through the hand-coded change, i.e., what specific elements of the resulting transactions are related to which specific features of the process bubbles in the requirements model. Maybe the design takes into consideration other requirements information which is not recorded at all! This is why the nicest initial software documentation tends to becomes useless after a few months of maintenance. The tracing model must therefore preserve detailed relationships between components as a basis for repeatability, with as few uncontrolled steps as possible.

The problem of traceability would be solved if we had a CASE tool that would offer a choice of both mapping methods for isA hierarchies of activity specifications. Upon arrival of complaints from the customer, the designer would simply revise his design decision to choose *method1* (map activities to transactions one-to-one) and activate *method2* (collapse the isA hierarchy so that transactions are created only for its leaves), as shown in figure 9. It is now clear that the two designs are design *variants* for the same requirements model.

This step deals with problems (1) and (2) of section 3. It also highlights another traceability requirement on CASE tools: not only the invocation of the tools but also important method choices *within* the tool must be made visible to the repository.

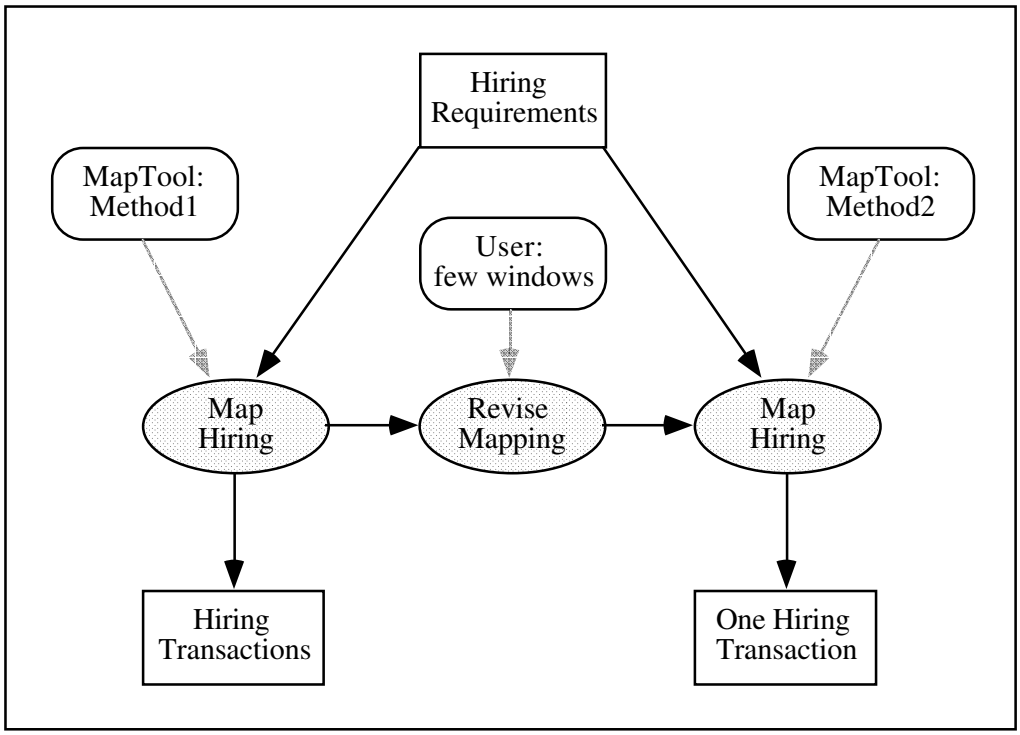

Fig. 9: Development process using CASE tool with multiple methods.

4.2 Decisions and Arguments: Capturing Rationale of Design Processes

When we just said "the designer would simply revise his design choice", this was clearly an over-simplification. In fact, the decision involved multiple stakeholders, especially the complaining customer who created the reason for the design decision. Of course, much dissatisfaction would be avoided if a quality team including customer representatives had detected the problem earlier (e.g., using design inspection and prototyping techniques) such that the problem could have been corrected before product delivery.

Besides this communication problem, there is also the issue of future systems maintenance, i.e., the problem of communicating across time. Numerous writers — e.g., Potts and Bruns (1988) — have therefore demanded recording not only the design decisions but also their rationales in the repository.

The demands of teamwork in the short run and rationale recording in the long run can be combined if rationale is captured in the form of argumentation records in the system. In figure 9, this is indicated by the user argument "few windows" given by the user to justify the decision to revise the mapping.

In the general case, a whole set of supporting and opposing arguments will be exchanged in design teams. These arguments may debate the contribution of different methods to a given nonfunctional requirement, the relative importance of different requirements in a particular design decision, and — most importantly — they might bring up new issues and requirements to be taken into account in the design.

Through recording this information, we have dealt with problem (3) of section 3 and achieved a *repeatable process* in the terminology of Humphrey. Systems like REMAP (Dhar

and Jarke 1988, Ramesh and Dhar 1992) demonstrate how modeling at this level technically supports repeatability by dependency-directed reasoning.

4.3 Decision Classes and Process Rules: Abstracting From Experiences

As a consequence, when users want to have a few additional input and output fields for the hiring transactions, the designer (maybe his successor by now!) simply adds the new attributes to the requirements model of figure 5 and repeats *method2* of the CASE tool to produce the new software version. Alas, he is unlucky again!

The customer complains that some fields of the transaction form have vanished from the screen. Closer inspection shows that the screen was simply too small. Obviously, the previous breakdown left another nonfunctional requirement undetected, namely, that a *size limit* must be taken into account when chosing a mapping method.

At this point, the team decides that repeatability is not enough: Total Quality Management demands a general rule that *defines* the process. The rule the team arrives at is "choose method 2 when the generated transaction has less than 10 attributes, else method 1".

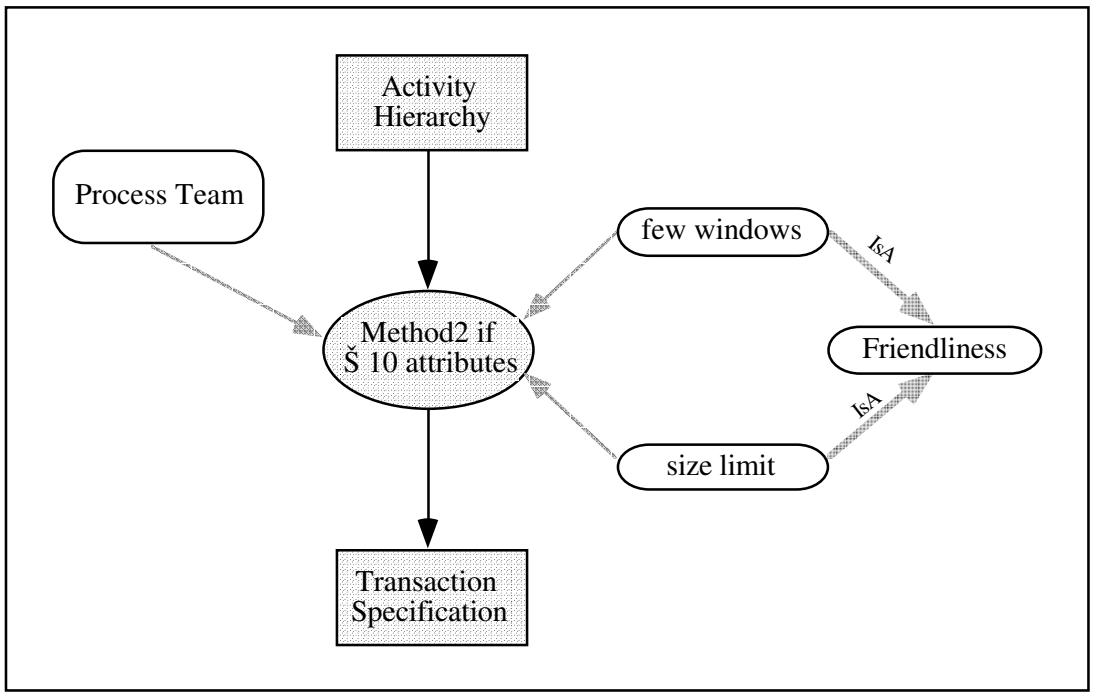

Fig. 10: Defining a process through quality-grounded decision rules.

To support such decisions, the repository technology needs the representational means to operate on the level of process definitions as opposed to individual projects. It cannot make do any more with generic classes such as "design object" "decision" or "argument". Instead, we have to see such models as *process meta models* under which refined class hierarchies are defined. Particular class hierarchies describe meta-objects such as functional requirements (design objects) and nonfunctional design goals (abstracted arguments) as well as methods (i.e., classes of decisions) to decompose, satisfice, and discuss these objects.

The generic process rule above would be defined as a class of decisions or method, as shown in figure 10. It operates on functional input of type *Activity Hierarchy* (i.e., isA

hierarchy of activity models at the requirements level) and produces output of type *Transaction Specification*. There are two methods available to execute this decision class; the choice rule among these methods was influenced by the trade-off between two nonfunctional goals (few windows and size limit) and decided by a certain group, the *Process Team*.

Several metamodels for describing this kind of quality oriented software process structure in a generic way are being developed in ongoing projects. Taken together, they resolve problem (4) of section 3 and lift our model to the maturity level of defined processes.

Structuring cooperative development processes. Jarke and Rose (1991) present the CAD^o metamodel as a formal framework in which object and goal classes, decision classes and the dependency types they generate, as well as agent teams and their conversational structures can be uniformly represented and related to each other. This model — shown in figure 11 — provides a generic framework for relating teamwork issues and process definitions. Using the classification abstraction of the underlying knowledge representation formalism, *quality assurance* can be automatically supported by requiring that each object and operation documented in the repository must be an instance of a defined object respectively decision class (Jarke et al. 1992). The formalism does not address directly the issue of how to represent specific quality-driven methods and their use in the design process, nor does it cover the capturing of exceptional experiences as a basis for process improvement.

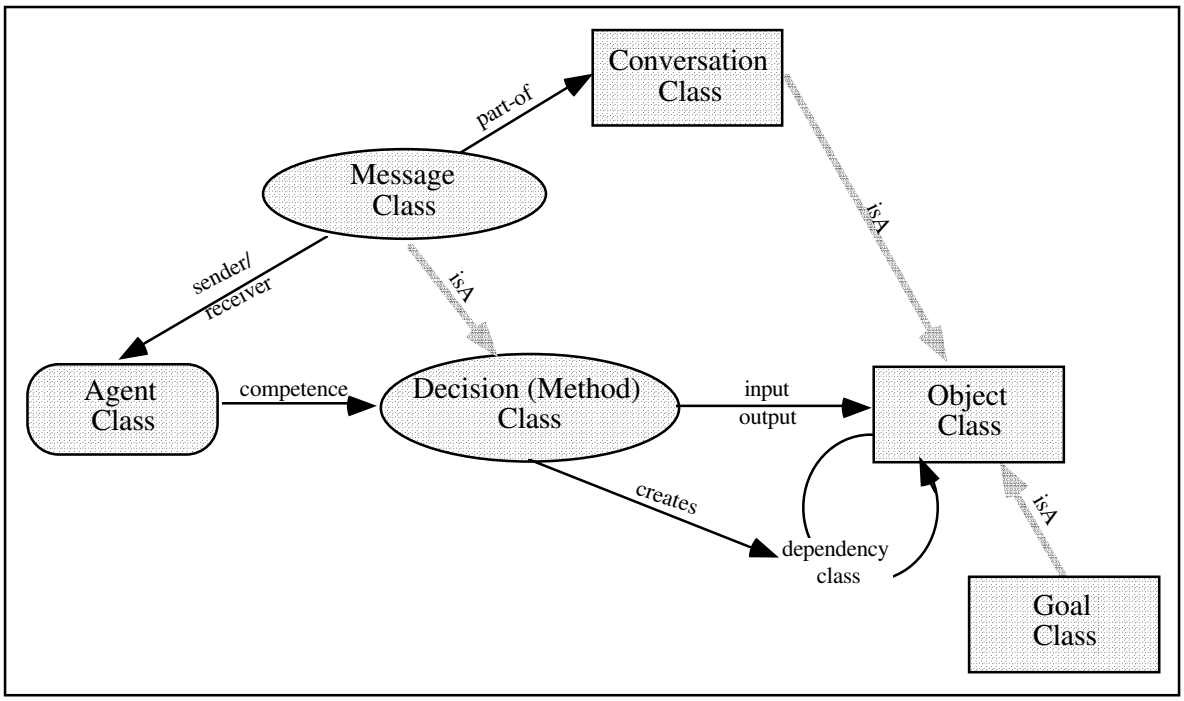

Fig. 11: CAD^o metamodel of cooperative design processes.

Organizing Quality-Driven Design. Figure 12 is a semantic network representation of a model proposed in a search-tree formalism by Mylopoulos et al. (1992) for representing and using nonfunctional requirements in the design process. Figure 10 can be understood as an example application of this model. It distinguishes 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 — in figure 10, the decomposition of user friendliness into "few windows" and "size limit".
- Satisficing methods describe how to achieve these goals — in our case, the two mapping methods.
- Argumentation methods define the relationships between subgoals and methods — in our case, the fact that the two methods exclude each other and that there are arguments for both of them, thus requiring a human decision.

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; therefore, 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 quantitatively expressed multi-criteria quality aspects. Both methods are mostly geared towards producing quality by applying the right methods, rather than measuring the actual quality of applying the defined rules.

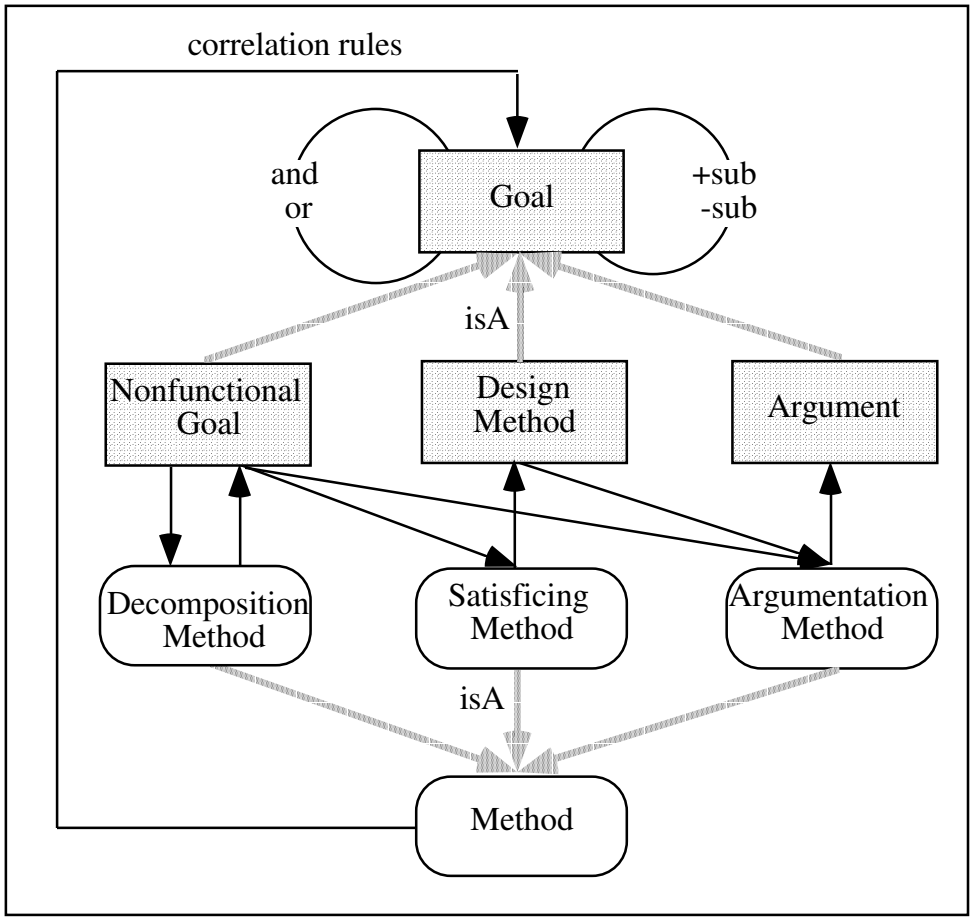

Fig. 12: Metamodel for quality-driven design process.

Domain analyses for specific quality models. A quality-oriented process metamodel achieves its potential only when instantiated with detailed knowledge about specific quality

factors and methods for their achievement. Domain analyses for specific kinds of functional and nonfunctional requirements create resuable guidelines and repository structures for process evaluation and management. Some expert systems have been developed that support for 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 et al. 1992) — 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).

Version and configuration management. Domain analyses do not only apply to methods for individual steps but also to information systems development in-the-large. Version and configuration management is not only important at the level of individual project histories but also at the metalevel of process definition (see next subsection).

As an extension to the DAIDA framework, Rose et al. (1991) have developed and prototyped a comprehensive domain analysis of conceptual-level *version and configuration management*. The versioning decision of figure 8 would be classified as being of type revision, whereas the one of figure 9 would be classified as a variant decision. The semantics of these two decision classes can be exploited for configuration process management. From the fact that the initial design mapping created a consistent configuration of requirements and designs, the repository can infer in figure 9 (but not in figure 8) that the variant configuration which combines the requirements with the second design is also consistent. This is the formal reason why requirements changes in the former setting are much easier to propagate than in the latter.

4.4 TQM Support: Structure for Process Improvement

Now that we have the means both to document and to define information systems development processes, we get to the real meat of the TQM approach: how to improve process definitions based on process observations — the managed and optimized levels of Humphrey´s hierarchy combined with teamwork. Unfortunately, the repository literature has not really addressed this question yet, nor do we have a fully satisfactory answer. However, we wish to point to some elements that will have to be part of the answer. Three elements are to be considered, each associated with one of the corners of the TQM triangle shown in figure 2:

- *process control:* how to relate process definition and observation
- *teamwork:* how to support the generation of process improvement by teams
- *management commitment:* how to document and manage process evolution.

Relating process definition and observation. There are several ways how feedback from process observations can be applied to process improvement. One is the identification of critical components and quality problems in individual projects (Selby et al. 1991). Another is the construction of a library of reusable process and quality models based on case observations

This latter approach is followed by the TAME project (Basili and Rombach 1988), with an emphasis to *quantify* and measure the software process. Their metamodel — shown in figure 13 — relates the well-known notions of goal and method to each other through the so-called GQM (Goal-Question-Metric) model in which a set of questions is first associated with each goal, and then a set of quantitative metrics to answer each question. These metrics are applied to evaluate executions of the methods in actual development processes, thus providing feedback for subsequent improvement. In contrast to the model of figure 12, the relationships between goals and methods are not explicit at the class level. As a consequence, the improvement process itself cannot be easily represented either. However, a rich model of reusable methods and GQM patterns is available. Yau et al. (1990) sketch an expert systems framework that elicits quality information about processes from designers based on such models, using the very similar factor-criterion-metrics model of Cavano and McCall (1978).

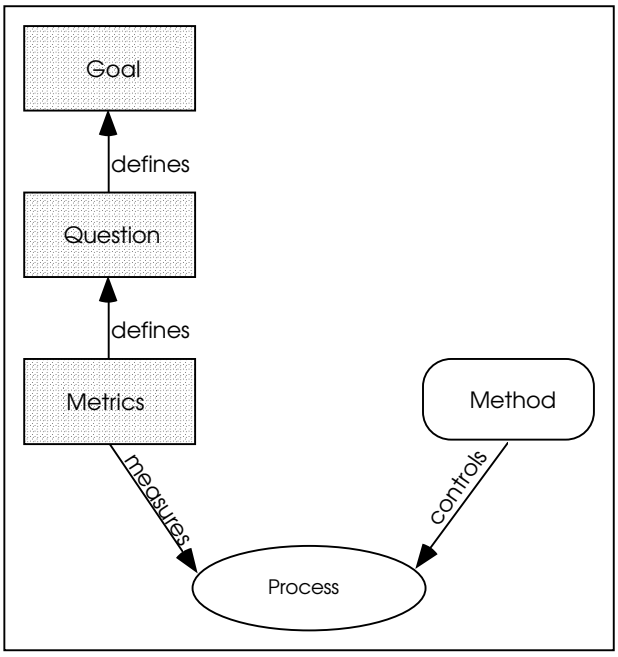

Fig. 13: Process measurement according to the TAME metamodel

An important prerequisite for observation-based process improvement are tools for the visualization of the joint effect of multi-dimensional quality effects. Figure 14 is a simple example of such a visualization. Each goal corresponds to a sector, its relative importance is shown by the width of the sector. The darkened area show how close the evaluated process or product comes to some "ideal" value for that goal. Specifically, figure 14 shows a comparison of the initial process rule given in figure 10 and an improved process (discussed at the end of this section) with respect to the goals of user friendliness, portability, and maintainability achieved by products created from these processes. It is obvious that user friendliness — considered most important — has not changed but that the new process improves the two other factors. In general, we would like to point out that the field of multi-criteria decision-making has a lot to contribute to the acquisition and presentation of such multi-dimensional goals (Jarke 1988). However, it does not really support the teamwork that develops and uses such models.

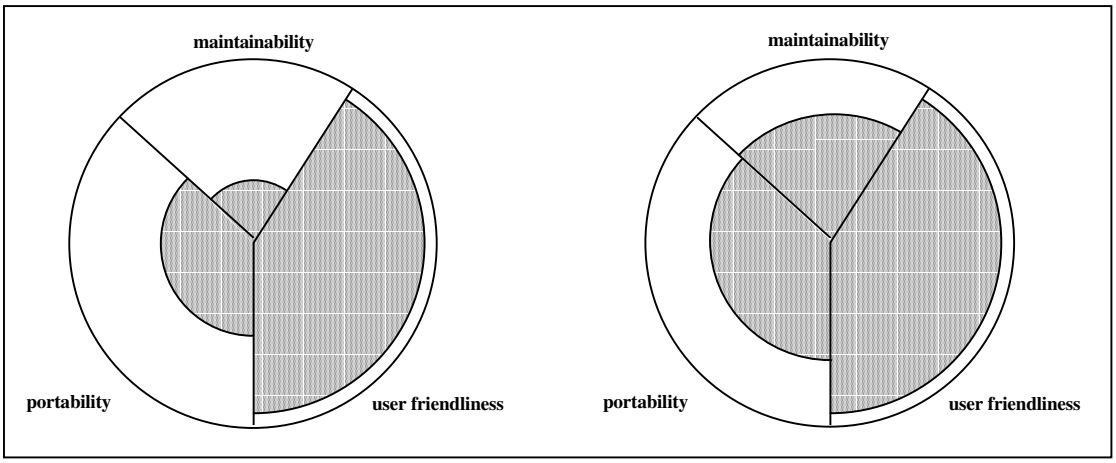

Fig. 14: Achievement of quality goals before and after process change

Teamwork support for process improvement. In order to be able to understand a group debate and to draw reasonable conclusions from it, it appears necessary to impose some structure on the argumentation process. Among several models that have been experimented with, Rittel´s Issue-Based Information Systems model (figure 15a) seems to offer the best coverage, provided it is adequately coupled with the other process information. In the gIBIS system (Conklin and Begeman 1988), this relationship is only in informal (hyper-)text form. Ramesh and Dhar (1992) therefore relate the gIBIS model formally to a structured documentation of requirements choices.

From a teamwork perspective, argumentation models such as IBIS support process improvement in that a collection of existing positions and arguments may suggest new issues, such as process improvement needs and opportunities. Figure 15b records an IBIS discussion about the process improvement that led the team from the repeatable process of figure 9 to the defined process of figure 10. In the terminology of TQM, the group considered the documented arguments as a sample of the software engineering process and generalized the successful and less successful process experiences to the process rule.

Thus, the IBIS model does not prevent discussion of process improvements. However, it does not offer specific operators for it either. It is completely up to the human quality team to decide whether the sample for this "statistical process control" is representative and what it means in terms of process improvement.

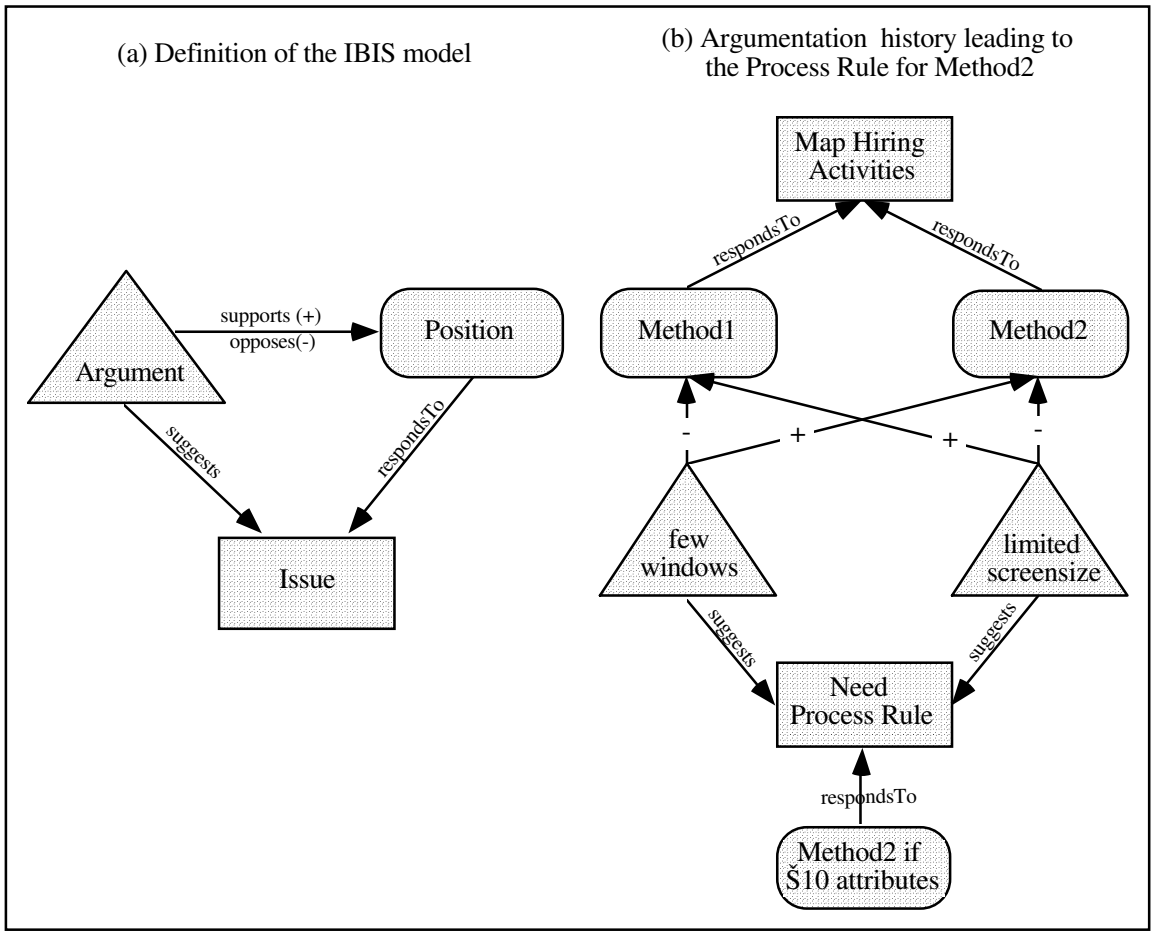

Fig. 15: IBIS model of argumentation: definition and application to process improvement.

Documenting and managing process evolution through meta-operators. Of course, management commitment cannot be replaced by repository technology. However, the repository can support management commitment to changes developed by teamwork through formal descriptions of typical kinds of process re-definition.

For the improvement process to be less haphazard than in the basic IBIS model, the repository has to be extended by metalevel operators which allow management to control and document process evolution. It is a major research project to identify a complete set of such operators, so we give here only some examples.

The process shown in figure 15b is a typical example for the *inductive generalization* operator. Such operators have been proposed for the learning of design rules from prototypes (Dhar and Jarke 1988), and for the learning of decision trees for analyzing measurements of software resource metrics (Selby and Porter 1988). They are also the reason for building up an "experience factory" in TAME (Oivo and Basili 1992).

The following refinement process shows another generalization operator, *parameterization* of process definitions. Suppose that our example IS has now become fairly successful so that it is ported to other environments. Some of these environments use workstations with much larger screens and application of the standard process rule generates too many windows again. The team therefore decides to parameterize the design rule with the screensize.

A third operator could be called *non-monotonic extension.* While the previous operators just generalize the earlier rule without invalidating it, one could imagine a workstation with a

large screen where user comprehension puts a limit on the size of windows and therefore on the applicability of method 2. The above rule could then be extended by "...except when there are more than 15 attributes on the screen". Borgida and Williamson (1985) employ machine learning techniques for refining integrity constraints based on the collection of such exceptions in the context of database schema evolution.

As a last example, consider the following case of *simultaneous engineering*. When the team stops to consider usage world and development world issues sequentially after one another, they suddenly notice that the simultaneous consideration of user friendliness and maintainability leads to a much simpler version of method 1. The transactions generated from upper-level activities (like Hire Employee) is called by the implementation of lower-level activities (like Hire Employee for Project) as a subroutine. There is thus only one requirements-to-design mapping procedure to be maintained. The issue of presentation on the screen is completely extracted into a generic user interface management procedure. Its application analogous to method 1 or 2 can then be managed under the goal of user-friendliness alone. Maintenance and configurability of the system are substantially enhanced. Of course, it depends on the openness of the CASE mapping tool whether this idea can be easily integrated, thus reiterating our statement that quality-oriented repositores will have to be complemented by better CASE tools.

Figure 14 compares the situation after applying the first operator (i.e., the initial rule) with the process as defined after the last improvement step discussed above.

5 Summary and Conclusion

This paper has identified a number of shortcomings in current CASE environments with respect to information systems quality management. Through a simple example, we derived a set of requirements to improve individual CASE tools and then focused on the question of how to organize their application through quality-oriented software information systems, for short: quality information systems.

The results of this analysis are summarized in table 3. They have been organized to match the SEI process maturity levels but emphasize teamwork support as in TQM over centralized control. Nevertheless, the models discussed show that formal and partially automatable representation and reasoning mechanisms exist for many tasks in quality information systems. At the same time, such a framework places stringent demands on the individual CASE tools to be integrated in it. Most importantly, the methods offered by the CASE tool must be visible and controllable from outside the tool, and the tool must be able to co-exist simultaneously with the repository in the same user environment. For the higher levels of the process maturity hierarchy, adaptability of the tool´s internal control mechanisms to the process definition as well as multi-user access to tools for improved teamwork must also be offered.

process maturity levels	repository concepts	problems addressed	requirements on CA
repeatable at project (instance) level	tools and methods dependencies decisions and rationales	traceability of processes 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 to 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 3: Summary of requirements for quality information systems and CASE tools

Many aspects of the proposed framework have still to be elaborated, especially concerning the human-machine division of labor and the requirements they place on repository organization and interface. From the representational viewpoint, monotonic abstraction mechanisms may not be enough to support intuitive understanding in large multi-criteria process models; we need organized over-simplification that 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; the repository must inform the team not only about the documented intentions and rationales but also about unintended side effects. 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 which is intended to address these issues in depth. In parallel, we are also applying and evaluating our repository model in a project on knowledge-based systems for TQM in industrial engineering we are jointly conducting with German engineering research centers.

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 Musa, J. D. "The Future Engineering of Software: A Management Perspective." IEEE Computer, Volume 24, No. 9, 1991, pp. 90-96.

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.

Boehm, B. W.; Brown, J. T.; Kaspar, H.; Lipow, M.; MacLeod, G. J.; and Merritt, M. J. *The Characteristics of Software Quality*, North-Holland, 1978.

Borgida, A.; and Williams, K. "Acommodating Exceptions In Databases And Refining The Schema By Learning From Them" In *Proceedings 11th International Conference on Very Large Data Bases,* August 21-23, Stockholm*,* 1985, pp. 72-81.

Bowen, T. P.; Wigle, G. B.; and Tsai, J.T. "Specification of Software Quality Attributes" *RADC-TR-85 37*, Volume I, II and III, Rome Air Development Center, Griffiss Air Force Base, February, 1985.

Bush, M. "Improving Software Quality: The Use of Formal Inspections at the Jet Propulsion Laboratory." In *Proceedings 12th International Conference on Software Engineering,* Nice, France, 1990, pp. 196-199.

Cavano, J.; and McCall, J. "A Framework for the Measurement of Software Quality". In *Proceedings of the ACM Workshop on Software Quality Assurance,*, 1978, pp. 133-139.

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.

Dhar, V.; and Jarke M. "Dependency Directed Reasoning and Learning in System Maintenance Support" *IEEE Transactions on Software Engineering,* Volume 14, No. 2, 1988, pp. 211-228.

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

Dijkstra, E. W. "Structured programming" In J.N. Buxton and B. Randell (eds), *Software Engineering Techniques*, Bruessel: Nato Science Committee, 1970.

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

Fagan, M. E. "Design and Code Inspections to Reduce Errors in Program Development." *IBM Systems Journal*, Volume 15, No. 3, 1976, pp. 182-211.

Fagan, M.E. "Advances in Software Inspections." *IEEE Transactions on Software Engineering*, Volume 12, No. 7, July, 1986, pp. 744-751.

Fenton, L. H. "Response to the SHARE software service task force report" In *IBM Corp.*, Kingston, NY, March 6, 1984.

Gilb, T. *Design by Objectives*, North-Holland, 1987

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.

Greif, I. (ed.). *Readings in Computer-Supported Cooperative Work,* San mateo, Ca: Morgan Kaufmann, 1988.

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.

Hausen, H. L.; Müllerburg M.; and Schmidt M. "Über das Prüfen, Messen und Bewerten von Software: Methoden und Techniken der analytischen Software-Qualitätssicherung" *Informatik Spektrum*, Volume 10, 1987, pp. 123-144 (in German).

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.

Jarke, M. "The Design of a Database for Multi-Person Decision Support". *Annals of Operations Research,* Volume 16, 1988, pp. 393-411.

Jarke, M. (ed.) *ConceptBase V. 3.0 User manual*, Report MIP-9106, University of Passau, Germany, 1991.

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.; Mylopoulos, J.; Schmidt, J.W.; and Vassiliou, Y. "DAIDA — An Environment for Evolving Information Systems", *ACM Transactions on Information Systems*, Volume 10, No. 1, 1992.

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, 1991.

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

Keller, S. E; Kahn, L. G.; and Panara, R. B. "Specifying Software Quality Requirements with Metrics" In R. H. Thayer and M. Dorfman (eds) *Tutorial : System and Software Requirements Engineering,* IEEE Computer Society Press, 1990, pp. 145-163.

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.

Motro, A. "Integrity = Validity + Completeness". *ACM Transactions on Database Systems,* Volume 14, No. 4, 1989, pp. 480-502.

Myers, G. J. *Software Reliability, Principles and Practices*. New York: Wiley, 1976.

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.

Norman, R. J.; and Nunamaker, J. F., Jr. "CASE Productivity Perceptions of Software Engineering Professionals." *Communications of the ACM*, Volume 32, No. 9, 1989, pp. 1102-1108.

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. 6, 1992.

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

Potts, C; and Bruns, G. "Recording the Reason 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.

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.

Sasso, W.C. "Motivating Adoption of KBSA: Issues, Arguments, and Strategies". In *Proceedings 6th Knowledge-Based Software Assistant Conference,* Griffiss Air Force Base, N.Y., 1991, pp.143-154.

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.

Selby, R.W.; and Porter, A.A. "Learning From Examples: generation and Evaluation of Decision Trees for Software Resource Analysis". *IEEE Transactions on Software Engineering,* Volume 14, No. 12, 1988, pp. 1243-1757.

Selby, R.W.; Porter, A.A.; Schmidt, D.C.; Berney, J. "Metric-Driven Analysis and Feedback Systems for Enabling Empirically Guided Software Development". In *Proceedings 14th International Conference on Software Engineering,* Austin, Tx, 1991, pp. 288-298.

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.

Stallman, R.M.; and Sussman, G.J. "Forward Reasoning and Dependency-Directed backtracking in a System for Computer-Aided Circuit Analysis", *Artificial Intelligence,* Volume 9, No. 2, 1977, pp. 135-196.

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.

Yau, S.S.; Wang, Y.-W.; Huang; J.G.; and Lee, J.E. "An Integrated Expert System Framework for Software Quality Assurance". In *Proceedings 14th International Computer Software & Applications Conference,* Chicago, Il, 1990, pp. 161-166.

Yourdon, E. *Structured Walkthroughs*. (3rd edn.), New York: Yourdon Press, 1985.

Zens, N. "Produktivitätssteigerungen in der Anwendungssystementwicklung durch CASE? " Diploma Thesis, University of Passau, 1990 (in German).

Acquisition Concern	User Concern	Quality Attribute
Performance - How Well Does It Function ?	How well does it utilize a resource? How secure is it? What confidence can be placed in what it does? How well will it perform under adverse conditions? How easy is it to use?	Efficiency Integrity Reliability Survivability Usability
Design - How Valid Is the Design ?	How well does it conform to the Requirements? How easy is it to repair? How easy is it to verify its performance?	 Correctness Maintainability Verifiability
Adaptation - How Adaptable Is It ?	How easy is it to expand or upgrade its capability or performance? How easy is it to change? How easy is it to interface with another system? How easy is it to transport? How easy is it to convert for use in another application?	 Expandability Flexibility Interoperability Portability Reuseability

Table 1: The RADC software quality consumer oriented attributes

(Keller, Kahn and Panara 1990, p. 146)

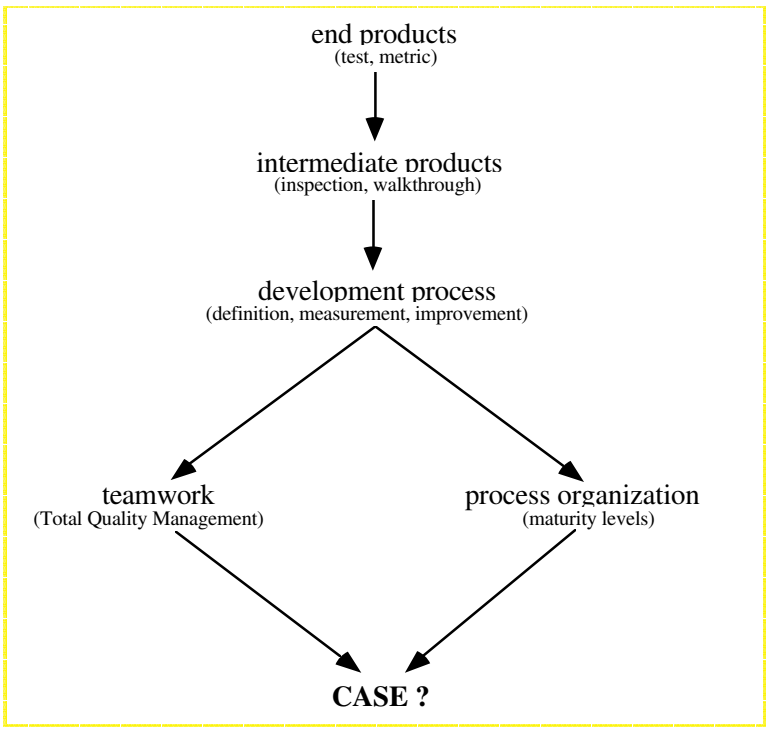

Fig. 1: Evolution of quality management in software environments

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

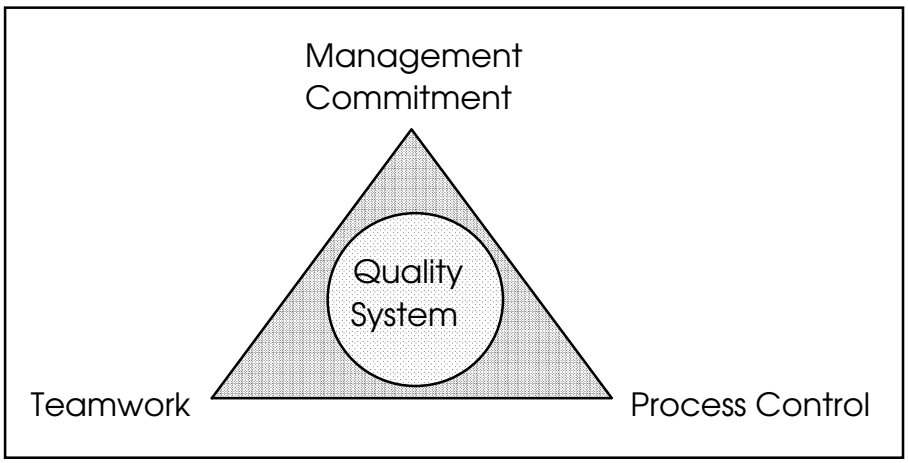

Fig. 2: The TQM triangle (adapted from Oakland 1989, p. 15)

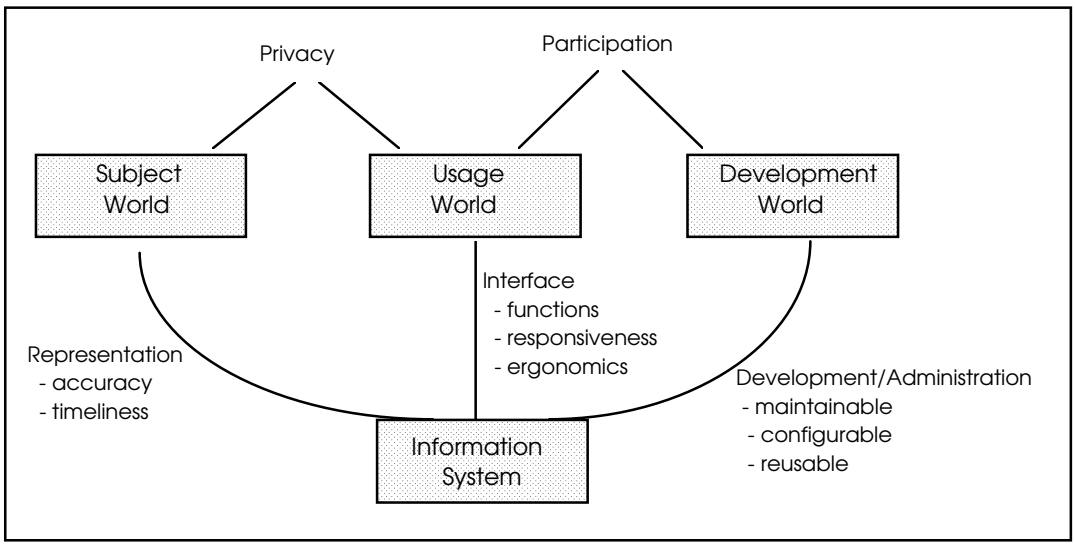

Fig. 3: Organizational principle for IS quality factors

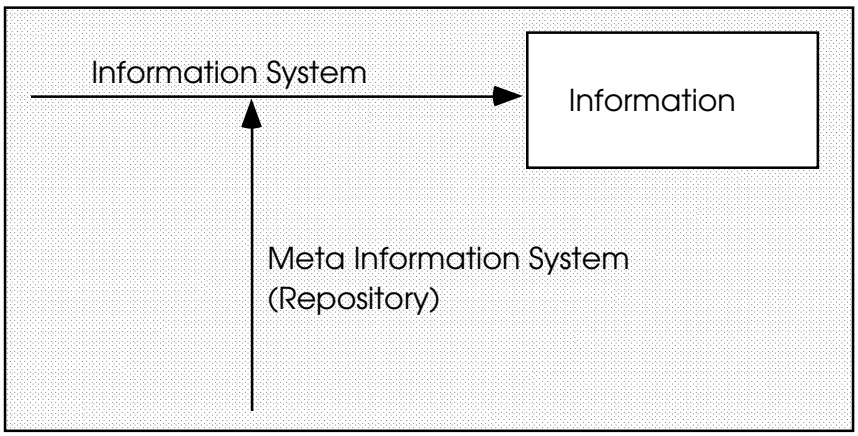

Fig. 4: Information system as a process, repository as a meta process

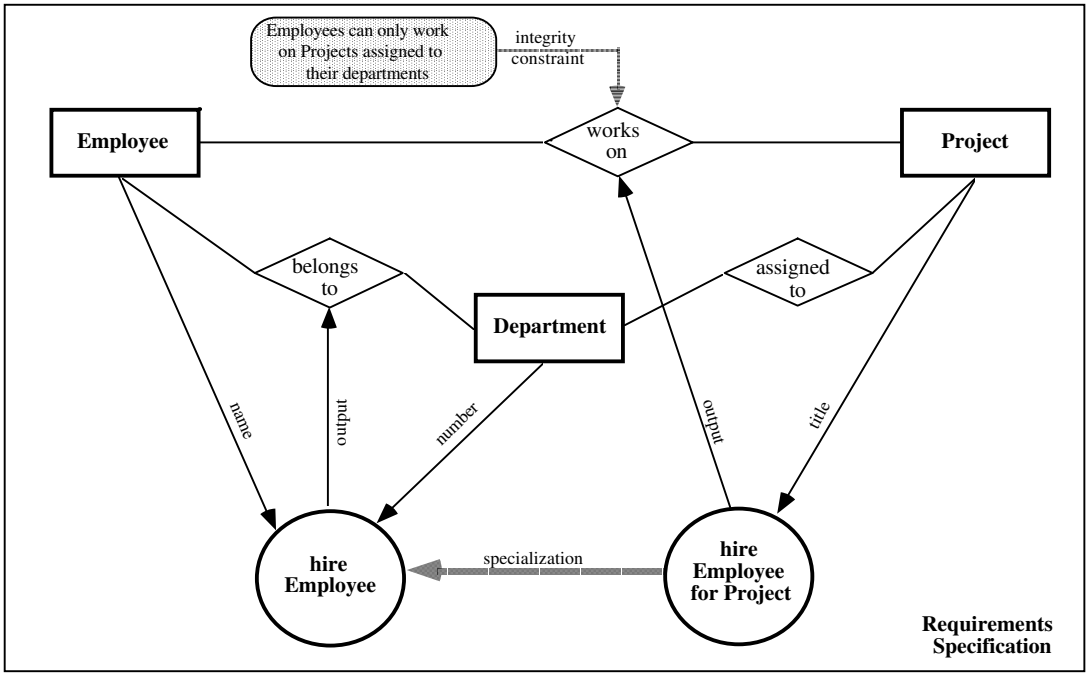

Fig. 5: Requirements specification of the example

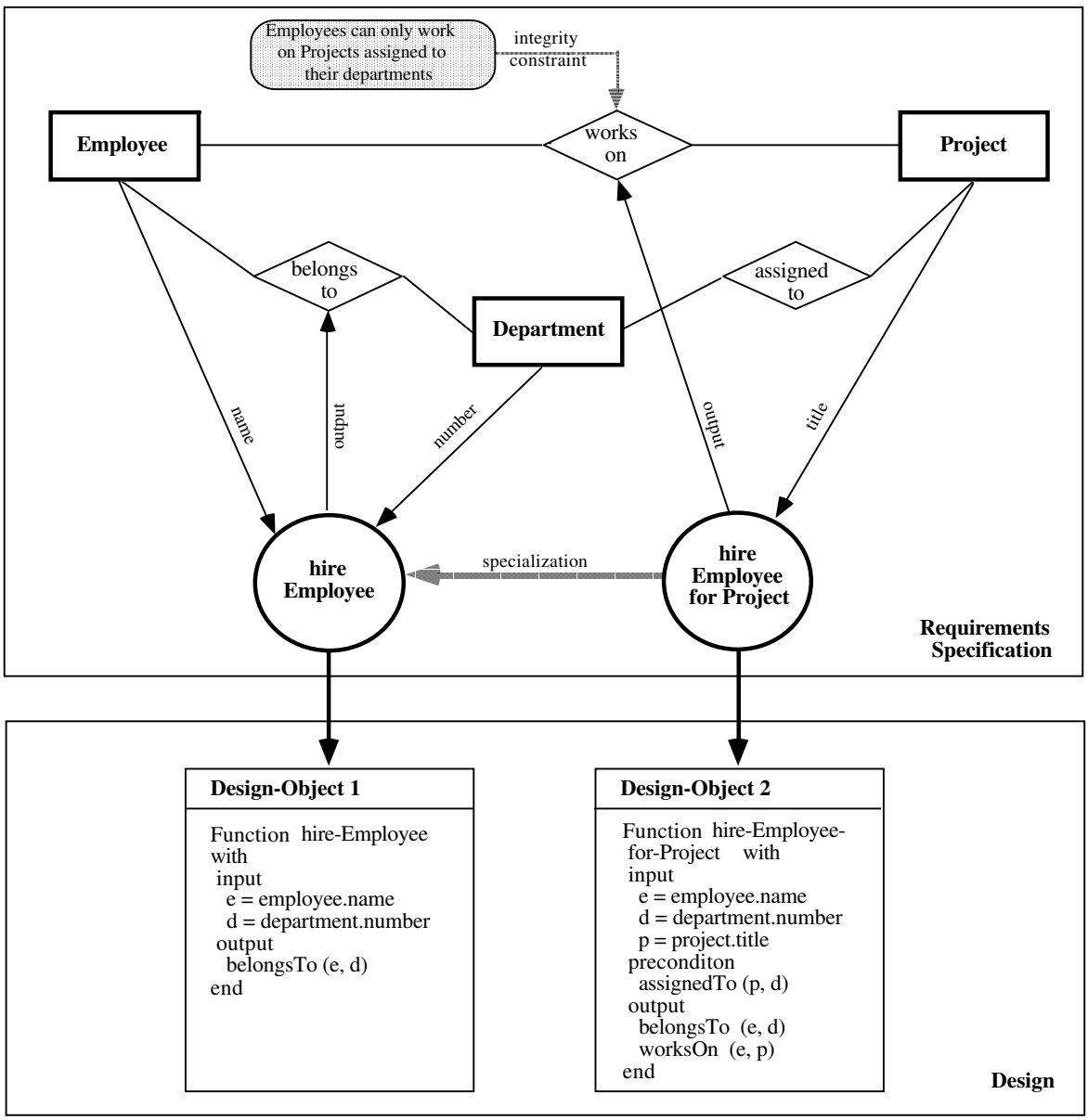

Fig. 6: Design generated from the requirements by the CASE tool

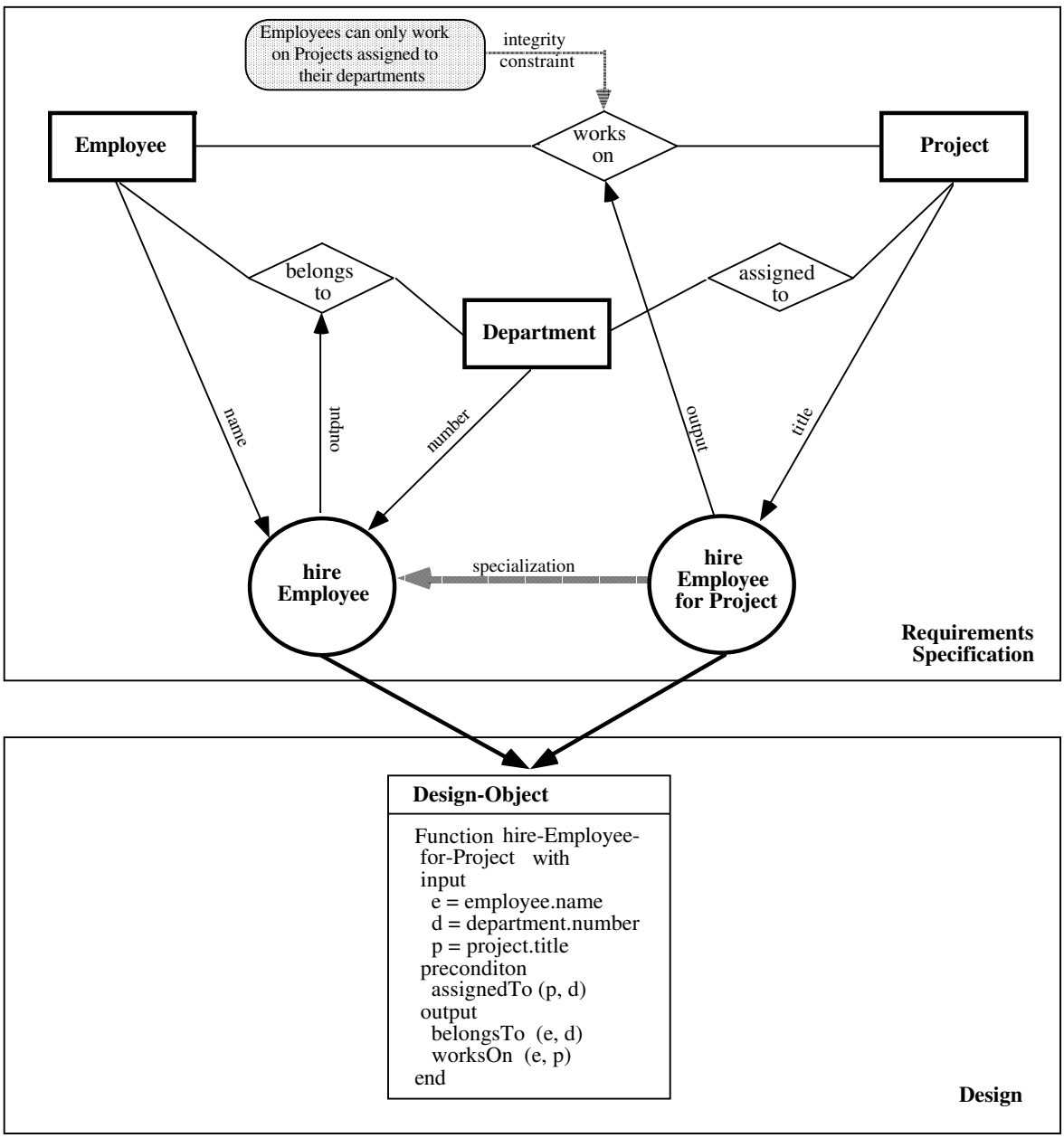

Fig. 7: The mapping solution after the change, as understood by the designer

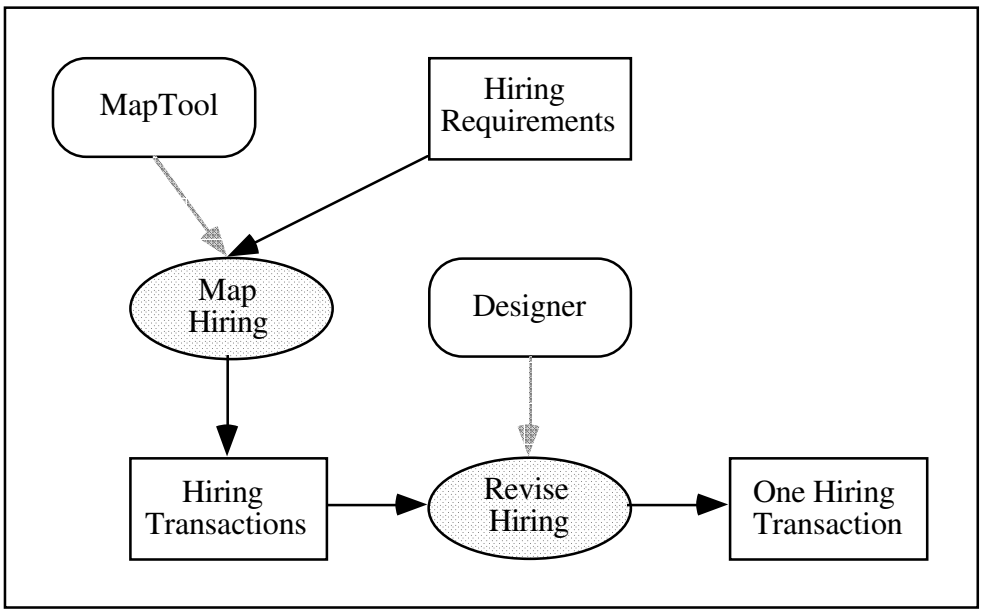

Fig. 8: Documentation of example development process

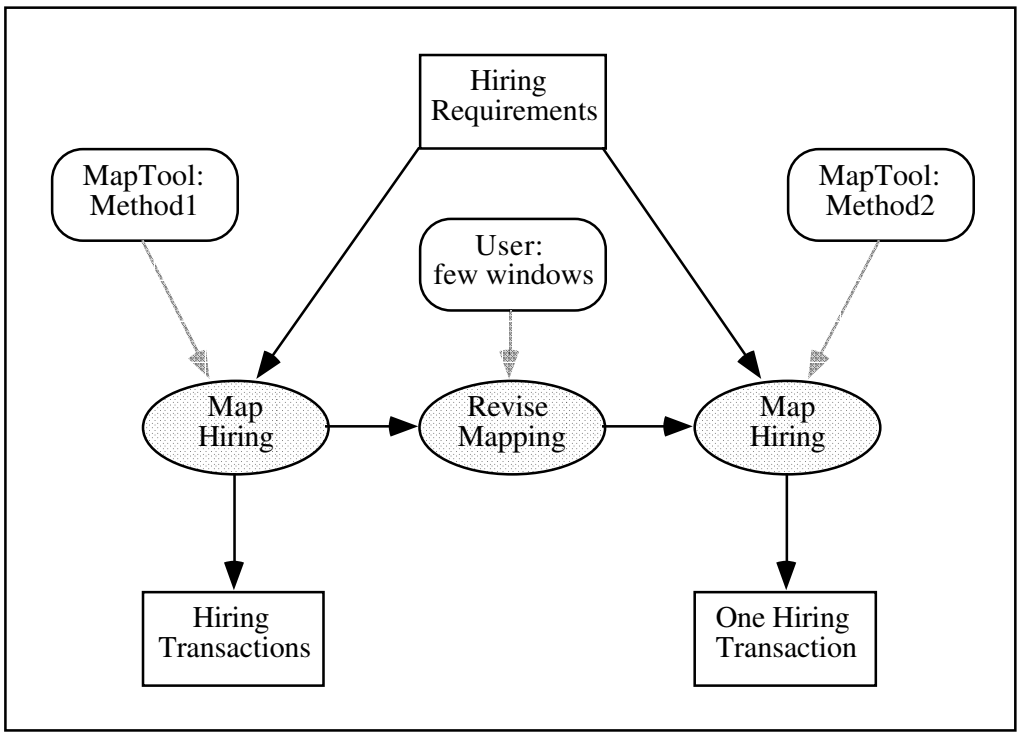

Fig. 9: Development process using CASE tool with multiple methods

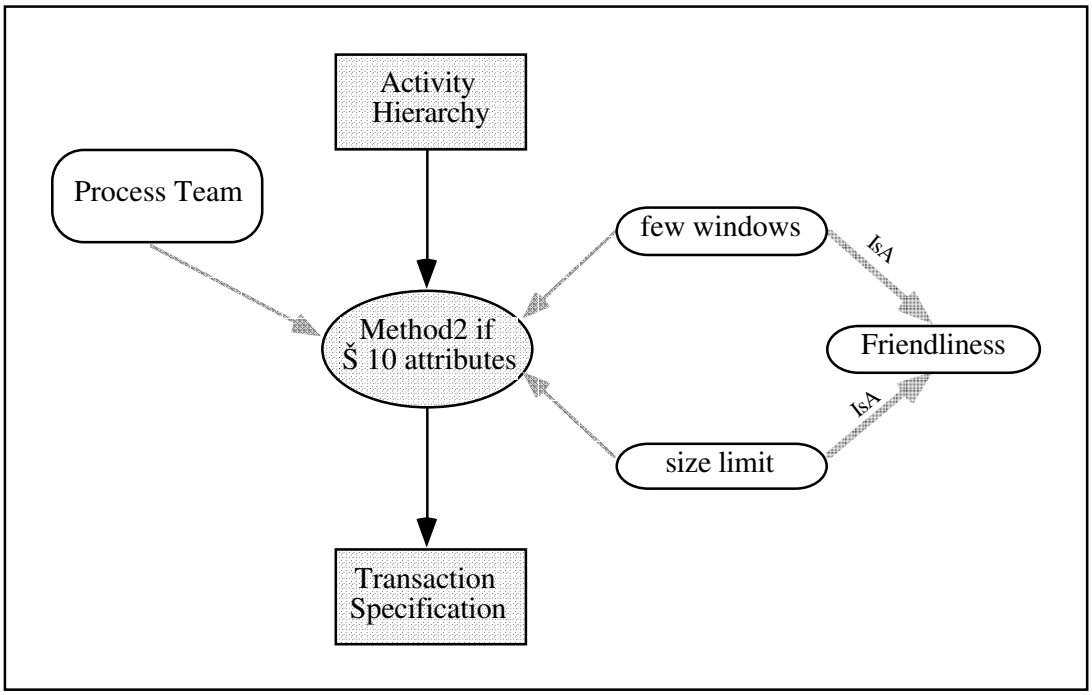

Fig. 10: Defining a process through quality-grounded decision rules

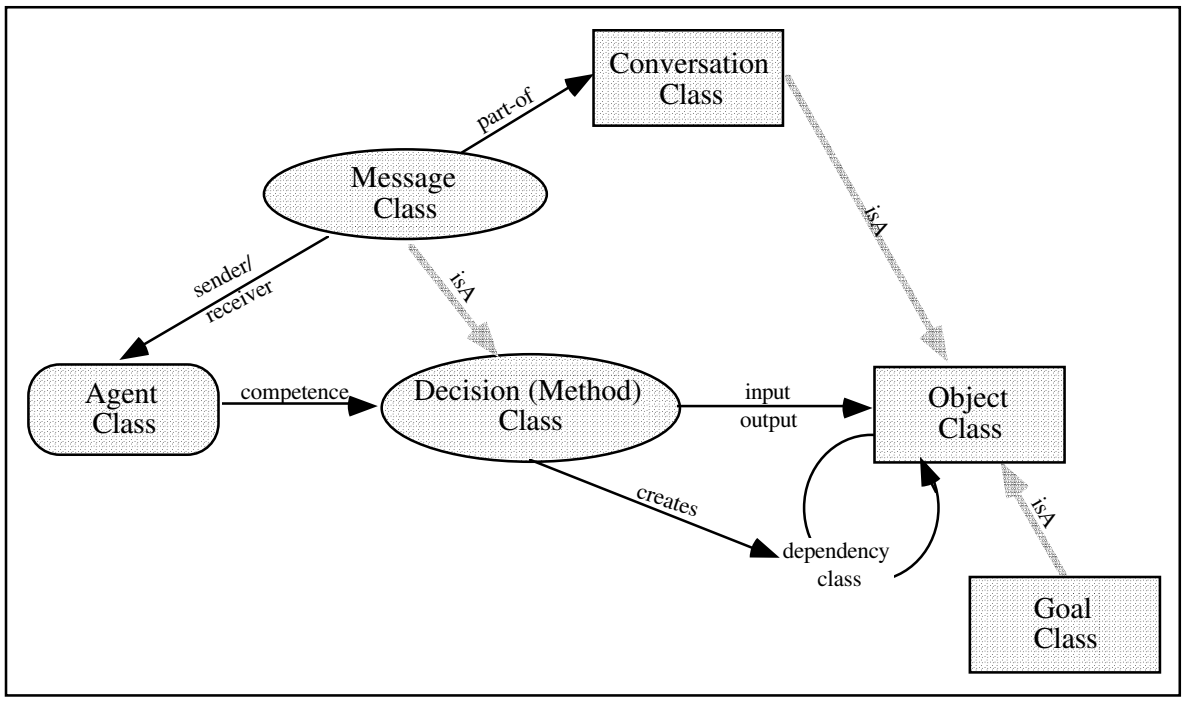

Fig. 11: CAD^{o} metamodel of cooperative design processes

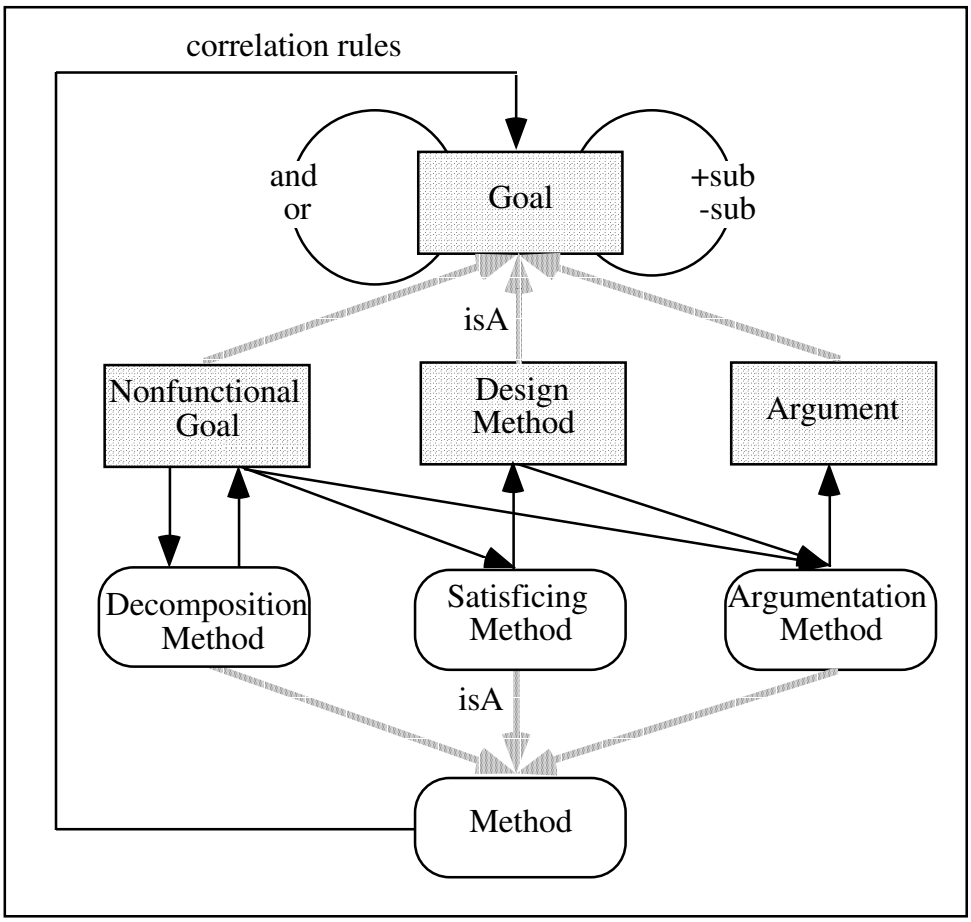

Fig. 12: Metamodel for quality-driven design process

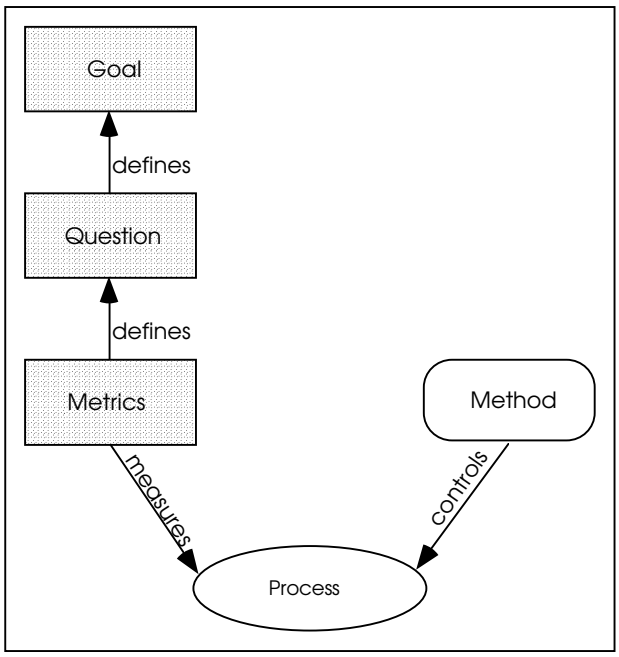

Fig. 13: Process measurement according to the TAME metamodel

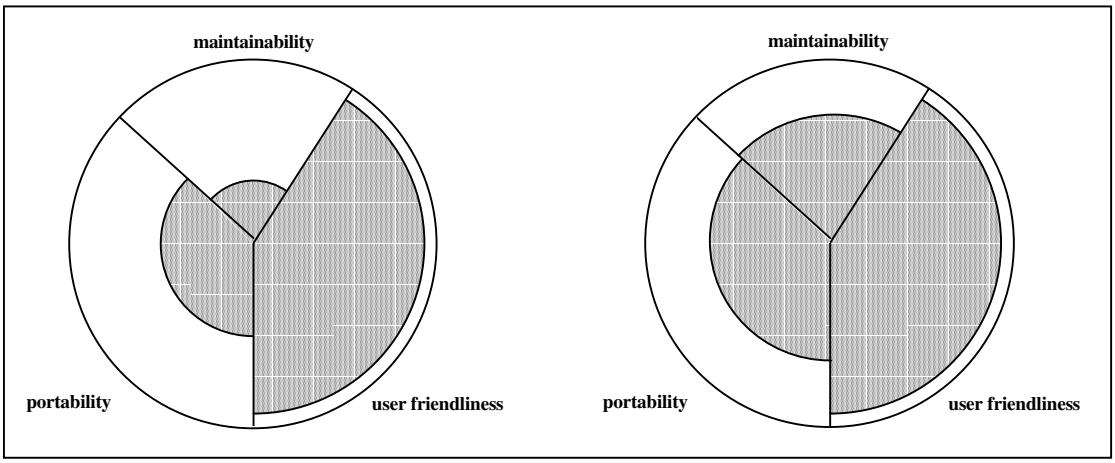

Fig. 14: Achievement of quality goals before and after process change

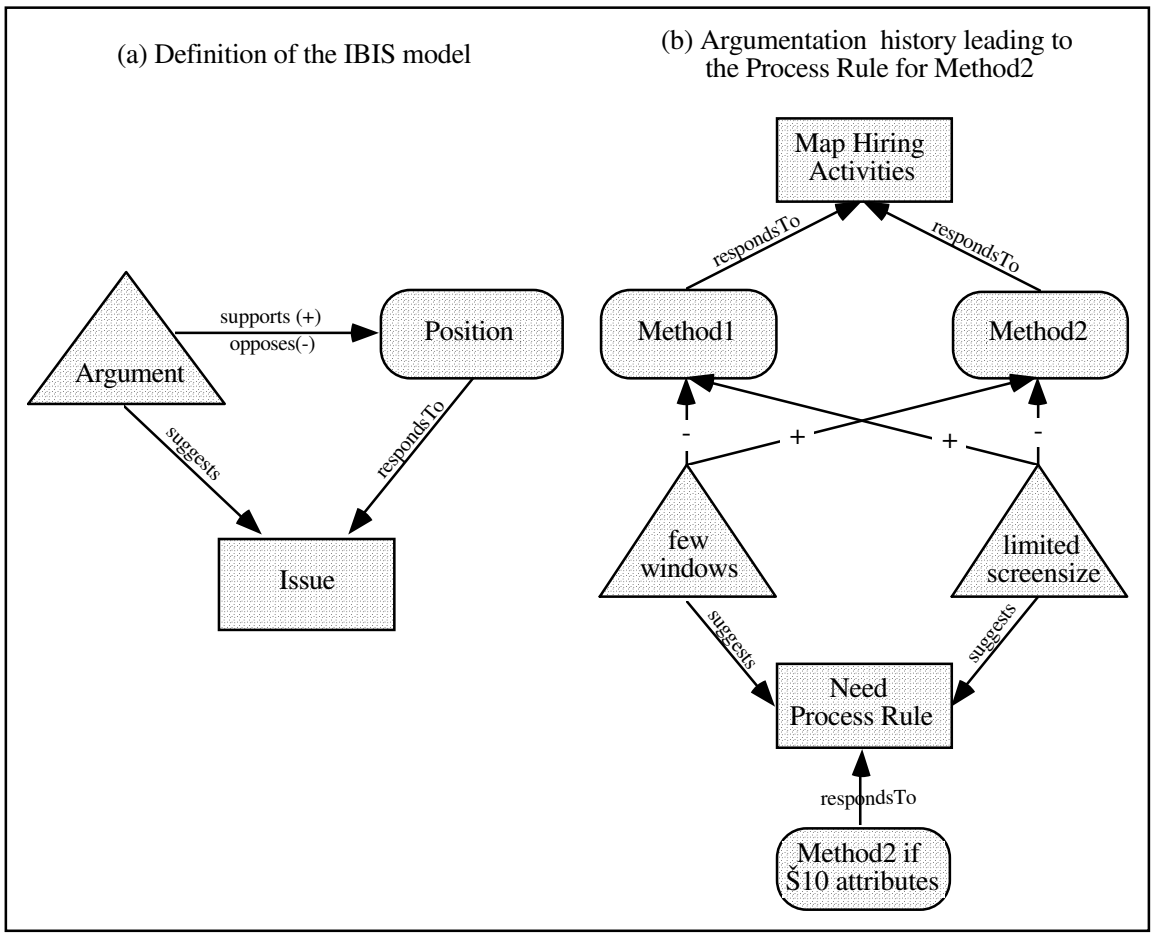

Fig. 15: IBIS model of argumentation: definition and application to process improvement

process maturity levels	repository concepts	problems addressed	requirements on CA
repeatable at project (instance) level	tools and methods dependencies decisions and rationales	traceability of processes 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 to 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 3: Summary of requirements for quality information systems and CASE tools