Journal of the Association for Information Systems

RESEARCH ARTICLE

Managing Evolutionary Method Engineering by Method Rationale*

Matti Rossi
Helsinki School of Economics
mrossi@hkkk.fi

Balasubramaniam Ramesh
CIS Department
Georgia State University
bramesh@gsu.edu

Kalle Lyytinen
Case Western Reserve University
kalle@po.cwru.edu

Juha-Pekka Tolvanen
MetaCase Consulting ltd
jpt@metacase.com

Abstract

This paper explores how to integrate formal meta-models with an informal method rationale to support evolutionary (continuous) method development. While the former provides an exact and computer-executable specification of a method, the latter enables concurrent learning, expansion, and refinement of method use (instances of meta-models) and meta-models (evolution of method specifications). We explain the need for method rationale by observing the criticality of evolving method knowledge in helping software organizations to learn, as well as by the recurrent failure to introduce rigid and stable methods. Like a design rationale, a method rationale establishes a systematic and organized trace of method evolution. Method rationale is located at two levels of type-instance hierarchy depending on its type of use and the scope of the changes traced. A method construction rationale garners a history of method knowledge evolution as part of the method engineering process, which designs and adapts the method to a given

* Matthias Jarke was the accepting senior editor for this paper. Jan Pries-Heje and Ralf Klamma were reviewers for this paper.

organizational context. A method use rationale maintains knowledge of concrete use contexts and their history and justifies further method deployment in alternative contexts, reveals limitations in its past use, and enables sharing of method use experience. The paper suggests how a method rationale helps share knowledge of methods between method users and engineers, explores how method engineers coordinate the evolution of the existing method base through it, and suggests ways to improve learning through method rationale.

Introduction

Information systems developers face an unprecedented pace of change as seen in the simultaneous rise of standardized object-oriented method (UML) (OMG, 2003), new visual programming environments, Web services, and new implementation platforms (e.g. mobile clients, distributed group technologies, and plug-in components). This has led to a situation where methods of yesterday provide a poor match for emerging development practices and radically new technical platforms. At the process level, increased outsourcing and new development contexts create unforeseen needs for method management and deployment. For example, Internet developers struggle constantly to invent and adopt practices that scale to the complexity, size, and agility of the business processes being supported as well as to the novel characteristics of development processes (such as speed, variation) (Rose and Lyytinen, 2003). Likewise, the construction of software-enabled products is increasingly supported by tailored method families that make possible rapid delivery of product variants[1] (Tolvanen and Kelly, 2000; Weiss and Lai, 1999). At the level of method use, method change forms a complex and reflective endeavor, where contents of the method, its justification, and its scope and style change continuously (Russo and Wynekoop, 1995). The abundance of UML variants is a prime example of constant method modifications that emerge as a response to new contingencies and diverse use experience (see e.g. Conallen, 1999; Desmond et al., 1998; Heberling et al., 2002).[2]

The criticality of adapting, developing, and maintaining methods and related computer-aided support environments is increasingly recognized as a crucial development process in its own right (Brinkkemper, 1996; Harmsen et al., 1994b; Hidding et al., 1993; Hofstede and Verhoef, 1996; Kumar and Welke, 1992; Odell, 1996; Tolvanen et al., 1996). This process -- denoted widely as *method engineering* (ME) -- is a (meta-level) systems development process applied to IS development practices and supporting tools. There is an increasing stream of study about how to engineer new methods (Floyd, 1986; Hidding et al., 1993; Odell, 1996; Russo and Wynekoop, 1995; Tolvanen et al., 1996), but there is a paucity of research on how methods in use evolve and can be

[1] For example Nokia and several other companies have used specialized methods to specify and manage functionality in software for phones that allows rapid generation of new phone models (Kelly and Tolvanen, 2000, Tolvanen and Kelly, 2000)

[2] These modifications are also becoming increasingly critical for effective system development so that tool vendors now offer extensible CASE tools to accommodate such modifications. For example, Microsoft will offer as a part of its forthcoming .Net CASE-tool functionalities that make possible non-standard UML extensions. (Randell and Lhotka, 2004)

supported with and by method engineering. Yet, as noted, methods change incrementally in response to changing technical and organizational contingencies (Floyd, 1986; Hidding et al., 1993; Russo and Wynekoop, 1995). In light of this, we suggest that *method engineering needs to be analyzed as a continuous, evolutionary process that supports the adaptation of methods to changing technical and organizational contingencies and new development needs*.

In this paper we analyze a particular facet of evolutionary method engineering. We demonstrate that a continuous stream of new methods or method variants and their explicit justification can become a crucial means for organizational memory and learning in system development. Recording method changes and their reasons enables and promotes learning and knowledge sharing within the community of designers, especially in rapidly changing environments. We call a trace of evolutionary method changes and associated use experiences *a method rationale* (MR). Method rationale involves both organizational and technical management of a set of dependencies across method use situations and method (meta) models.

The specific goal of this paper is to explore how organizations can build up and benefit from an explicit method rationale that can support evolutionary method engineering. We show how method rationale leads to new ways of applying methods and provides a systematic means to communicate and learn from method use. We illustrate method rationale construction and use processes by analyzing a concrete example that has been implemented with a metaCASE environment (MetaEdit+). It exemplifies building and using a method rationale when adopting and implementing a new standard version (e.g. 2.0) of UML and its extensions into a CASE tool.

The remainder of the paper is organized as follows. In the next section we motivate a method rationale first by demonstrating the inevitability of method change in two areas of method research: process improvement and agile development research. In the next section we define method development --method engineering (ME) -- and identify weaknesses in the past ME research in how it has viewed method knowledge and its changes. In section 4 we offer a sketch of an evolutionary ME process and discuss the architecture of a method rationale and related tool support. Section 5 exemplifies how a method rationale can be implemented and used in a metaCASE environment when designers consider adopting a new version of the UML method. In the last section we discuss remaining research challenges in developing truly reflective and evolutionary support environments that can improve fast-changing development practices.

Inevitability of Method Change

Development methods and processes and related capabilities have been long regarded as valuable assets of development organizations. Optimization and effective reuse of methods and processes can significantly enhance development productivity and quality (Holdsworth, 1999). At the same time, software development is not a standardized manufacturing process in that the same process can be repeated for every aspect of a project outcome as well as across multiple projects. For example, methods used in developing even successive versions of the same software vary considerably. The need to find a good fit between specific project circumstances and generic methods has been recognized as one key challenge in software development (Cockburn, 2000; Karlsson et

al., 2001; Kraiem et al., 2000; Zmud, 1980), as building methods from scratch for each development situation is risky and creates significant overhead. Therefore, existing methods are normally adapted to meet the increasingly varying needs. There are several different contexts where this challenge has been recognized, including two extremely topical areas: process tailoring and agile methods.[3] An exploration of these areas will justify our call for managing better method change and offering a more systematic way to do it through evolutionary ME and method rationale.

Process Tailoring

Process tailoring can be defined as the "act of adjusting the definition and/or particularizing the terms of a general description to derive a description applicable to an alternate (less general) environment" (Ginsberg and Quinn, 1995). The need for process tailoring has been identified in the "post modern" view of IS development (Baskerville et al., 1992), which argues that human organizations are emergent and continually adapt to a set of goals. Consequently, development processes will be unique and emergent in light of the constantly shifting requirements that confront an organization. Though standardized software process models are regarded to represent "best practices" within the industry, some of their elements do not meet the shifting requirements of emergent processes. Therefore, a host of elements have to be constantly tailored to meet the goals of each project (Ramesh and Jarke, 2001).

Ginsberg and Quinn (1995) made two empirical observations that support this view. First, they found that similar projects require different levels of tailoring due to differences in the organizational structure. For example, different contractors for the same government agency have to adjust their processes differently due to organizational differences. Second, they found that a single organization may contain environments having significantly different characteristics, and therefore it needs to employ different development processes. Hence, almost every organization or project must carry out tailoring in order to apply effectively "best" standard practices. A similar result was observed in a detailed study of traceability practices in large scale software development efforts (Ramesh and Jarke, 2001). The study highlighted the need to adapt practices to suit the varying needs of projects and organizations. The authors argue that organizations need to explicitly represent the conditions under which various process steps are executed in order to enhance the reusability and tailorability of these processes. They also illustrated the need of tracing method and process changes in order to improve the effectiveness of reusing system development processes. In light of this, Ramesh and Jarke propose a set of reference models to organize design rationale at different levels of granularity to depict contexts in which software development artifacts and processes are created and used.

The necessity of tailoring has also been recognized in multiple process improvement

[3] We could have adopted nearly any area of method use and deployment but these were chosen based on the suggestions of reviewers and the SE. For example, a similar need for method change and adaptability has been recognized in studies of method use and adaptation in user experience reports (Odell, 1996), surveys (Russo and Wynekoop, 1995, Tolvanen et al., 1996), and in methodological discussions of method research (Fitzgerald, 1991). This can be also generalized to the specific relationships between plans and situations as studied by Suchman (Suchman, 1987).

frameworks. Because all standardized software development process models such as ISO/IEC 12207 (IEEE/EIA, 1998a; IEEE/EIA, 1998b; IEEE/EIA, 1998c), IEEE/IEA 12207 (IEEE/EIA, 1998a; IEEE/EIA, 1998b; IEEE/EIA, 1998c) and RUP® (Kruchten, 2000) have been developed with a broad scope of situations in mind, they are too generic to be readily applied "as such" in a specific software project. As noted by Ginsberg and Quinn (1995), the SW-CMM model they studied defines only a generic set of practices that reflect "best" organizational practices of organizations that develop large software systems for government agencies. Yet, in order to appropriate these practices, organizations must significantly tailor them prior to their application (Ginsberg and Quinn, 1995). As they note, these generic practices provide only a fruitful starting point for improving software development processes. In fact, a growing amount of recent process improvement literature suggests that standardized methods are never adopted faithfully or followed rigorously (Holdsworth, 1999). Rather, their specific elements are selected and tailored flexibly to suit specific project needs. Such tailoring activities include, typically: eliminating unnecessary elements from reference models, adding new elements, and/or changing workflows,[4] For example, ISO 12207 and IEEE/EIA 12207 standards dedicate a whole section to specify activities and tasks for tailoring them for a software project or an organization. Tailoring standardized processes also offers several benefits. It reduces delays, increases productivity, and improves quality (Hollenbach and Frakes, 1996). It also helps transfer method knowledge between projects, and thereby reduces training and planning costs (Holdsworth, 1999).

At the same time, empirical studies of process improvement show that process tailoring is difficult in that it involves intensive knowledge generation and deployment (Demirors et al., 2000; Ginsberg and Quinn, 1995; Machado et al., 1999; Polo et al., 1999a; Polo et al., 1999b). Guidelines associated with managing and adopting process models are not detailed enough to guide developers through the (meta) process, even though a number of studies have proposed sets of factors that will influence process tailoring, including: domain characteristics, project characteristics, project goals and assumptions, organizational structure, corporate size, maturity level, etc (Ginsberg and Quinn, 1995; Holdsworth, 1999; Machado et al., 1999). For example Holdsworth (1999) extended the Capability Maturity Model (CMM) by adding an inventory phase to evaluate the current state of the project and the organization (taking stock of such variables as product portfolio, clients, suppliers, and staff). He also suggested mechanisms for identifying and tracing resulting modifications in the software development process (Holdsworth, 1999). On the downside, Holdsworth does not propose any systematic theoretical base for directing or managing such change. Another downside in process tailoring is that it demands large amounts of work to make the methods fit (Demirors et al., 2000; Polo et al., 1999a; Polo et al., 1999b). To address this, Avrilioni and Cunin (2001) have proposed the OPSIS approach to effectively reuse process assets. Their approach matches component interfaces with the process parameters and checks the consistency of the resulting processes. Karlsson et al. (2002, 2001) proposed a method to adapt software development methods by configuring a standard process model. Under their methodology, when a project's characteristics matched one of the recurring patterns of

[4] For example, Demirors et al. (Demirors et al., 2000) tailored ISO/IEC 12207 for instructional software development in small software development organizations. Polo et al. (Polo et al., 1999a) tailored ISO/IEC 12207 for maintenance. Machado et al. (Machado et al., 1999) applied ISO/IEC 12207 and CMM model to improve processes for service development.

project characteristics, it could employ a predefined process configuration. This approach seeks to create reusable process configurations based on experience from earlier projects (Karlsson, 2002, 2001). The approach leads to questions about whether each task/activity/step should be performed as it is, reduced, skipped, or extended. Yet they do not suggest any clear mechanisms for how such knowledge is obtained, managed, or shared as part of the process modification method.

Agile methods

The need for changing and adapting methods and managing related knowledge is not only necessary for organizations that employ formal/mature software processes, but also for those that use agile methods. In fact, the importance of the adaptability of methods has been increasingly emphasized when organizations have started to move to agile development. Proponents of agile methodologies that address needs of high velocity software development claim that traditional methodologies cannot be tailored to new environments, are bureaucratic, and therefore slow down the pace of development (Highsmith, 1999). In contrast to 'heavy' or 'monumental methodologies', 'light-weight methodologies' need to be "adaptive rather than predictive" (Beck and Fowler, 2000). Unlike heavy methodologies that plan in great detail over a long span of time, light methods "adapt and thrive on change, even to the point of changing themselves." In this environment, the ability to tailor and evolve a method across projects, and even across various phases of a project, becomes critical (Baskerville et al., 2001). Examples of this trend abound: Agile Software Process (ASP) addresses the accelerated pace of software development within geographically distributed teams by maintaining flexibility. Similarly, Conallen's (Conallen, 1999) methodology refines, extends and conflates the Rational Unified Process (RUP) (Kruchten, 2000) and the ICONIX process (Rosenberg and Scott, 1999). Likewise, Rising and Janoff (2000) claim that Scrum[5] can only be effective in projects where requirements are evolving and "chaotic conditions are anticipated throughout the product development life cycle."

By recognizing this trend, method developers for agile environments increasingly emphasize the tailorability of their methods to provide a "customized roadmap to development success" (Kruchten, 2000). However, adaptation and tailoring of methods is challenging for high velocity software organizations. Even with the flexibility of a tailored methodology, there is no guarantee that a stable process can be established. Even during the development of successive versions of the same software product, the methods followed can vary widely (Ramesh et al., 2002) depending on the composition of the project team and the nature of the product. Exceptions are made for team members with similar software development experience, due to the (time) criticality of the software, and the expected life span of software. Managers allow experienced designers to skip detailed designs or reviews as they have an established track record of quality output. As the products and development organizations mature, the methods have to evolve to meet the new demands for quality and stability. For example, when a product has matured and attained critical market share, the demands for quality (e.g., scalability, robustness, security, etc.) increase (often, dramatically), requiring changes in

[5] This is a method that uses small teams working on time-boxed development. The description fits Netscape's development effort for its early browser, where rapidly evolving requirements necessitated changes in the development methods throughout the product development (Iansiti and MacCormack, 1997).

the development methods. A major difficulty faced by developing organizations is that there are very few knowledgeable and experienced developers. Research suggests that most designers lack experience, and thus their ability to establish 'home grown' development methods, or to tailor existing methods, is limited. Research also suggests that when an organization gains experience in the use of ‘new’ methods, its ability to adapt and tailor them improves. In fact, such ability contributes toward increased product development agility (Thomke and Reinertsen, 1998).

Need for managing method change

The review of these two streams of method literature suggests that continually identifying and carrying out method changes in development environments is necessary in software development as it provides significant benefits, including:

- Improved performance, predictability, and reliability of methods,
- Faster and easier adoption of methods through better training of project personnel,
- Improved adaptability and agility of methods to changing needs of the development,
- More scalable, transferable, and measurable software development practices,
- Increased control of software development, in that teams consistently apply methods that help achieve more consistent outcomes, and
- Improved communications among team members.

Next, we shall look at how these benefits can be achieved. We suggest that software development organizations need to implement an integrated environment for method rationale that records method changes and their reasons. The environment involves both organizational and technical management of a set of dependencies across method use situations, and method (meta) models and can thus be best maintained with a metaCASE tool that supports evolutionary method engineering.

Method Engineering and Method Rationale

System development and method development = method engineering

Two separate domains of inquiry can be identified in information systems development: the problem system and the problem-solving system (Checkland, 1981). We call the problem system the object system under development, or the target system. This system embodies the technical sub-system being built the supported business system, and its environment. The problem-solving system consists of components that enable developers to identify, design, and carry out changes in the object system. One part of this is the development support environment, which consists of a set of tools, mental frames, and notations that are enacted during the change process (Lyytinen et al., 1989). Modeling languages and associated methods are an important element of the environment that dictate when, how, and by whom a notation like UML Class Diagram is to be used. Thus, *IS methods* provide a means and an environment for linguistic communication and technical problem-solving that encompasses the use, nature, content, context, and form of signs included and processed in the problem system, the information system (Lyytinen, 1987). A method can be defined as a set of techniques (e.g. procedures, associated possibly with a prescribed notation) to carry out a

development activity. A combination of these techniques is called an ISD method.

When the problem-solving-system is regarded to be an object of inquiry, we call this activity method development, or method engineering (Kumar and Welke, 1992). We depict the relationships among these different levels of inquiry in Figure 1. Method engineering provides methods and processes to specify, make explicit, codify, and communicate method knowledge, as well as technical tools to enact such processes effectively. These two inquiry processes are normally separated in time and space so that methods are just imported from a separate generic problem-solving environment and installed into the problem-solving system. After Orlikowski (1996), we call such a situation time-space disjuncture in method engineering. Most traditional method engineering literature assumes a sharp time-space disjuncture (Harmsen et al., 1994a; Kumar and Welke, 1992; Tolvanen and Rossi, 1996) and assumes a "one shot" blueprint method engineering approach where “method cowboys” ride into the organization, diagnose current problems, and develop a method and its process instructions that match with the observed contingencies (Harmsen, 1997). Thereafter, designers follow the steps enlisted in the manuals, or for the better, enact automated steps specified into a CASE tool. Yet in practice, as shown above, these two processes co-evolve, are intertwined and interdependent, resulting in evolutionary method engineering processes.

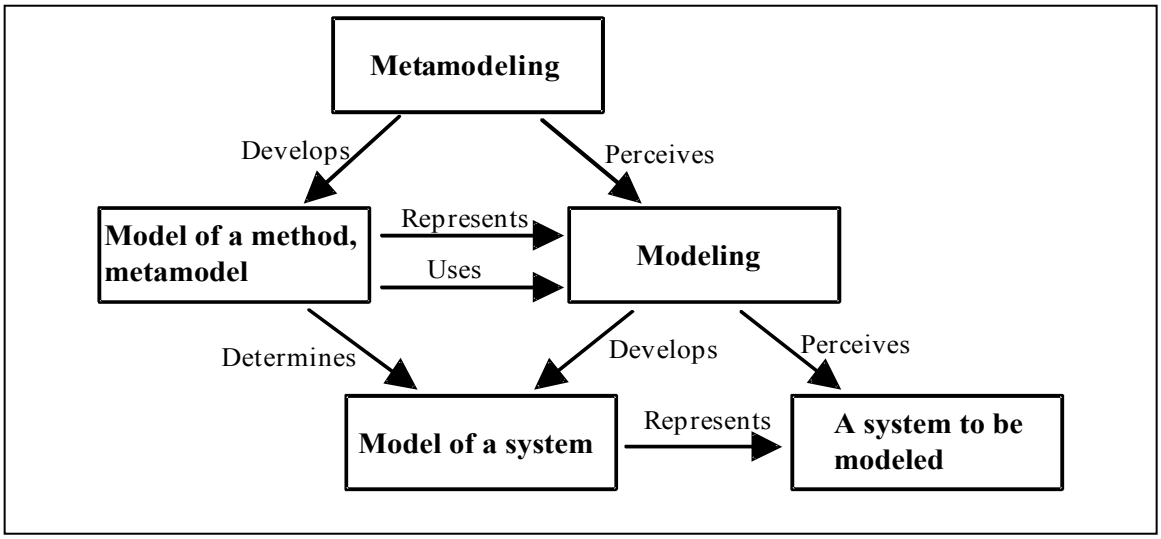

Figure 1 The relationships between modeling and metamodeling

In order to model methods, we need a set of concepts that can capture the content and form of a method into a *meta-model* (see Figure 1). In its simplest form, a meta-model is a conceptual model of a method (Brinkkemper, 1990), and exhibits a type-instance relationship between the meta-model and its instantiation, the IS model. This can also be seen as the development of a domain ontology or a set of domain ontologies for the target domain (Jarke et al., 1998). Consequently, meta-modeling can be defined as a modeling process within the problem-solving system, which takes place on one level of abstraction and logic higher than the primary modeling process (Gigch, 1991). Every modeling process in the problem-solving system implies a meta-modeling process because a meta-model captures concepts and representation forms that are necessary for the use of a method as part of the problem-solving system. How to formalize some parts of the knowledge within the problem-solving system into a set of formal method

specifications and how to make such method knowledge explicit, consistent, and systematic has been a subject of intensive research in method engineering (Tolvanen et al., 1996). In contrast, knowledge about decisions and processes that leads to a specific meta-model -- the situation analyses and the warrants for such decisions -- a *method rationale* has not been explored due sharp time-space disjuncture.

When method changes within the problem-solving system are slow, or when the time-space disjuncture can be maintained without any difficulties (i.e. the method users understand how to use the method without understanding all the contextual knowledge that leads to this form of method knowledge), the need for explicit method rationale is relatively low and relates to some changes in the problem-solving system (e.g. turnover of developers). The situation is different when meta-model change is continuous due to rapid changes in the problem-solving system, or due to its fast internal learning that results from trial and error processes (with little or no time-space disjuncture). In such situations, meta-models are constantly modified and extended in response to emerging needs and learning outcomes, leading to evolutionary method engineering. This increases the need to track changes in the meta-models *contextually* within the problem-solving system. Formal meta-models and their informal “change histories” help collect, organize, and analyze experiences related to the performance of the problem-solving system. These changes increase the "fit" of the method to a given situation in the problem-solving system, and ease its learning and applicability (i.e. the method overcomes with less friction barriers related to time-space disjuncture). This requires that explicit formal meta-modeling and method engineering must be extended with a more explicit concept and implementation of method rationale in the method development environment. In the following, we clarify first the idea of formal “one-shot” method engineering that maintains a sharp time-space disjuncture, and then contrast this with ideas of a reflective development practice leading to the notion of contextual and evolutionary method engineering, which lowers or removes the time-space disjuncture leading to incremental or agile method development. In the latter, the need for a method rationale becomes pronounced.

Blueprint Method Engineering and Evolutionary Method Engineering

The dominant approach underpinning ME and most ISD methods can be characterized as what Schön (1983) calls “technical rationality”: situations in practice can be scientifically categorized, problems are firmly bounded, and they can be solved by using standardized principles (Tolvanen, 1995). The main goal of method development in this *blueprint* view of ME is to provide complete knowledge about systems development that can be deployed without friction in the problem-solving system. Moreover, this knowledge is explicit and well defined. This goal, however, has been difficult to achieve due to the complexity and emergent nature of system development situations. Therefore more “relaxed” versions of blueprint method engineering have emerged. An earliest example of the increased situation dependency of the method choice was the formulation of contingency models that included, among others, the HECTOR project (Savolainen et al., 1990), Davis’ contingency model for requirements determination (Davis, 1982), or the Euromethod (Franckson, 1994).

During the 1990s FRISCO conferences dealt extensively with method matching and

adaptation (Falkenberg et al., 1996; Lindgreen, 1990; Stamper 1990). At the same time, within the method engineering community, situational methods received increased attention. For example, Harmsen's proposal for "situational method engineering" (Harmsen et al., 1994b) assumes that the developer first garners detailed knowledge about the situation (i.e. the needs of projects and organizations), and thereafter decides on the applicable method, choosing it from among a set of alternative method "frames". Recently, Ralyte and Rolland (2001) have proposed a meta-model for method fragment reuse based on capturing knowledge about the application domain and design activity. Similarly, Kraiem et al. (2000) and Punter (1996) have proposed similar meta-models for situational methods, including "contingency factors" that characterize a project and its environment. These approaches expect that all necessary knowledge about the method, either tacit or explicit, is made available during the method development phase. Methods embody enactable routines for development organizations, and therefore frequent method use is assumed to lead to repeatable processes. The key problem for method engineers is to select the right method rather than being concerned about how designers actually use it.

As the experience reported above shows, there is very little evidence that software development will become routine in, for example, a manufacturing process. In contrast, it will remain similar to the field of architecture,[6] where situational knowledge about the design is necessary and aids designers to gradually build appropriate design strategies organized around formalized method knowledge. Hence, method engineering involves a situated learning process in which the current level of expertise and the situation influence the use outcomes (Hughes and Reviron, 1996). During system development, such learning takes place at two levels; in the domain of a target system (IS), and in the domain of a problem-solving system (ISD). The former denotes learning about successful (or unsuccessful) ISs and their domains. The latter connotes that any organization that builds ISs not only delivers systems — they also learn how to carry out system development and to mobilize associated knowledge (methods), and as a result know how to improve their problem-solving system (learning by doing).

Hence, during ISD process, an organization -- or rather its members -- gains experiences about the applicability of the method, its use situation, and how it "talks" back to the user. These experiences complement the formal method knowledge the organization already possesses, and lead to new insights of the method's applicability. To our knowledge, only some learning-based approaches to method development (Checkland, 1981; Mathiassen et al., 2000; Wood-Harper, 1985) identify the importance of experience and learning from method use as a key mechanism that helps evaluate and refine methods. Early on, Checkland (1981) advocated a learning-based approach to method development by introducing a cycle of action research in which the experience of method use provides a main source for method modification. In light of this cycle, evolutionary ME can be viewed as a continuous and never-ending process, in which experience is continually elicited from the method use.[7] Unfortunately,

---

6 Note the significance of this analogy, as there is both standardized architecture as in the design of a mall or an apartment house, and the designs of great architects like Le Corbusier, Aalto, Gaudi, Lloyd-Wright or Gehry.

7 This view parallels with the goals of process improvement movement that have focused on improving the repeatability and optimization of processes by learning by doing and continuous

organizations often fail to record and reuse their internal experience. As Lyytinen and Robey argue (1999), a majority of this experience is lost because development experiences are never collected and interpreted. They live only as "war stories" that are narrated as part of the organizational culture (Orr, 1990).

In real situations it is never possible to have full knowledge about the problem system (and thus the applicable method), nor can pre-defined methods ever cover all possible situations. Furthermore, part of the methodical ISD knowledge remains tacit and cannot be fully specified. Therefore, development efforts that are carried out by faithfully following a set of pre-defined methods are impossible and, if followed, would be doomed to fail. A better way to understand the role of methods, while at the same time honoring our incomplete knowledge of ISD situations, is to view them from the organizational learning perspective. This perspective analyzes system development situations and the role of methods through a lens of what Schön (1983) calls “reflection-in-action.” In Schön’s theory, each situation is unique, and developers draw upon the tacit and experiential nature of their knowledge that emerges as the situation “speaks” to the developer (Nonaka, 1994). Accordingly, part of a designer’s knowledge of ISD is a result of his or her reflections of the situation, rather than being determined by predefined methods (transmitted formally to him or her through time-space disjuncture). In real situations, true “working” methods are appropriated and interpreted by designers based on their reflections of the situation. At the same time, methods are outcomes of those reflections in that designers’ tacit understandings are made explicit so that they can be conveyed to others (Nonaka, 1994).

ISD is complex and depends on the influence of many stakeholders. Accordingly, designers cannot develop systems by merely drawing upon their experience-based and tacit knowledge. Reflection-in-action and technical-rationality are complementary in that both explicit and tacit knowledge are necessary to understand system development. A good method that talks to the situation should cover both aspects: it should provide cognitive frames and norms that designers can use as a resource (Suchman, 1987), but it should also invite them to use their experiential knowledge (Argyris and Schön, 1978). This inspires them to expand their experiential knowledge and make it explicit. When designers adopt such a learning view toward method development, we call it evolutionary method engineering (Tolvanen, 1998).

Because evolutionary ME aims to continually improve ISD methods, it can be regarded as a learning process in which individuals (Schön, 1983), communities, and organizations (Nonaka, 1994), create, memorize, and share knowledge about system development (in methods) and how to apply it.[8] It involves a process of double-loop learning in which “error is detected and corrected in ways that involve the modification of an organization’s underlying norms, policies and objectives” (Argyris and Schön, 1978)[9]

---

feedback (1988).

[8] This is quite similar to Curtis et al. (1988) who suggested that both the developer and user learn through the dialectic approach.

[9] Similarly, (Floyd 1987, Fitzgerald, 1991; Oinas-Kukkonen, 1996; Schipper and Joosten, 1996) advocated a second-order learning process in which past experiences guide the use of the method. Her early emphasis on learning was important, because it allowed us to motivate and contextualize method rationale better.

leading to continuous modification and augmentation of an organization's methods.

Design Rationale in Method Engineering= Method rationale

Typically, an evolutionary ME process includes the following steps: problem definition, model formulation, model solution, model interpretation, and model maintenance. Thereby, evolutionary ME forms an iterative process that involves a second-level level loop of modification, elaboration, and refinement of methods. Operational scenarios, requirements, and assumptions that underlie methods evolve, necessitating continued reformulation of methods. This process is normally error prone and involves a significant amount of rework. This rework can be decreased by effective organizational memory that traces critical method decisions and records information about trade-offs that were made during method choices. Therefore, it is essential to capture process knowledge about the development and evolution of methods (i.e., method rationale) in order to facilitate their effective use and evolution.

In light of this, *we define method rationale as a characteristic of evolutionary method engineering in which the methods are linked to their intellectual sources (backward traceability) and to the method engineering outputs created during a meta-model life cycle (forward traceability)* (Ramesh and Jarke, 2001). We distinguish method rationale from design rationale (Ramesh and Jarke, 2001) in that the focus of method rationale is on capturing the "design" rationale behind ME artifacts, rather than ISD artifacts. Formally speaking, a method rationale system can be defined as a semantic network of dependencies where nodes represent conceptual and physical objects of the ISD domain and its context as they are continually produced during the ME process. A method rationale trace is established through different types of semantic links that connect diachronically or synchronically specific meta-model elements and their instances.

Through establishing such dependencies, a method rationale documents the reasons for the specific evolutionary steps in method knowledge that lead to the subsequent creation, modification, and alternative uses of the method. For each step in the method evolution, the trace can:

- Offer justifications for the creation and modification of methods,
- Record important decisions and assumptions,
- Identify the context in which method objects are created,
- Provide transparency into the decision process, including a trace of discarded alternatives, in order to provide a thorough understanding of the current solution,
- Facilitate maintenance and reuse by providing access to the history and context of different ME objects, and
- Manage method development in line with organizational needs and objectives.

Due to their "one-shot" focus, most current ME approaches fall short in addressing the maintenance, justification, and analysis of different method versions. Typically, only the final (in the true meaning of the word) product of the ME process is documented, but not the process of arriving at the solution. The only level where rationale enters the discussion is at the level of modeling decisions. A variety of design rationale approaches have been proposed to capture and reuse rationale behind critical design decisions (Lee, 1993; Ramesh and Dhar, 1992; Ramesh and Sengupta, 1995). The aim of these

approaches is to record and organize design decisions made using a predefined schema (meta-model) to represent the context in which they were made (Ramesh and Dhar, 1992). Hence these approaches focus on the decisions behind designs (e.g., the selection of one of the several possible ways to create an inheritance hierarchy in an application domain), but not on the decisions behind methods -- for example, why an inheritance between two classes is defined as virtual. Not surprisingly, most of these approaches have developed separate meta-models of design decisions outside specific modeling notations and process models, which are normal products of ME. Suggested tools are standalone tools and are integrated into CASE tools only in an ad hoc manner (see Kaipala (1997) for a different approach).

When a design rationale is expanded to support evolutionary method engineering, the recorded method "design" rationale can be used at least in two ways: record decisions and experiences related to method use and trace decisions related to method construction. An example of the former could be an IS developer's justification on why the concept of multiple inheritance is used in some specific inheritance structures. Approaches of this type focus on decisions related to method use within the ISD process. For example, Jarke et al. (1994) proposed a traceability model for following processes defined in a process guidance model. In contrast, during the construction or tailoring of a method for use in this ISD, the decision to support multiple inheritance should be based on an elaborate consideration of the implications of the method component choice for possible implementation languages and target platforms. Method construction rationale in this decision refers thus to knowledge that is critical for understanding the appropriate use of the method in future.

Support for learning and change using Method Rationale

In evolutionary ME, method evolution is seen as necessary, as organizations have to deal with different method versions for different implementation targets and development contexts (as for example with UML (OMG, 2003)), introduce new method types (such as object-oriented methods) based on vicarious learning, expand existing methods based on trial and error learning, and abandon old methods (unlearning). For example, we anticipate that after the introduction of profiles and extension mechanisms into UML 2.0, organizations are likely to experience these situations almost on a continuous basis.

Basically, two different types of method evolution can be distinguished: those reflecting general requirements of changed technical and business needs (vicarious learning), and those relevant to the ISD situation at hand (learning by doing). The former relates to the general genealogy of methodical knowledge within the IS (method) community, and the latter with how these general evolutions are adapted into local situations and affect development practices. In addition, we can observe specific evolutions that have political and power implications. These are often related to the need for downward compatibility to a specific technology that dominates the market or utilizing an installed base for specific methods for method expansion and refinement (e.g. a move from OMT to UML).

To meet the diverse needs of these method evolution types, we need a general framework for evolutionary ME tools that can support method evolution by maintaining a method rationale. This will trace and justify changes in methods, as well as method development processes, and integrate emerging feedback from user situations for

double-loop learning (Hidding et al., 1993).

Current State-of-the Art in Method Rationale

In current practice, method rationale lies in the heads of the people who have developed methods and accompanying tools. Other stakeholders such as method users cannot easily contribute to method construction because of communication problems related to time or place disjuncture as discussed earlier.[10] Consequently, important parts of the method rationale are typically lost and therefore, local method and tool adaptations can only be done partially and in ad-hoc fashion. One reason for this is the lack of tool support for collecting, integrating, and organizing method rationale systematically, which leads into ignoring available experience (Lyytinen and Robey, 1999). While some researchers have formulated conceptual aspects of method rationale (see e.g. Tolvanen, 1998; Kaipala, 1997; Ralyte and Rolland, 2001), no comprehensive analysis of the requirements for tool support has been made. Moreover, the rationale should be captured and integrated in a designer's natural use-context (Fischer et al., 1991). This implies that the capture of the evolving method knowledge must be supported with method engineering and design tools that are available in the CASE environment.

Overall, method rationale and method use experiences can be represented at different levels of formality. The most informal approach considers rationale as a free form textual annotation (e.g. Reeves and Shipman, 1992), which can be indexed for retrieval and analysis. Formality of the design is increased rationally in approaches where it is modeled as an argumentation process organized into specific discourse structures (e.g. IBIS (Conklin and Begeman, 1988), DRL (Lee, 1991), QOC (MacLean et al., 1991)) that are defined through appropriate semi-formal meta-models. In developed stand-alone design rationale tools (like gIBIS (Conklin and Begeman, 1988)), a fixed conceptual meta-model of the argumentation structure is represented as a directed graph. Additional functionality to analyze the argumentation structure includes manipulation of graph nodes and links (e.g. transitivity, weighting of arguments) and making queries of node contents across a graph. Unfortunately, most such tools like gIBIS do not adequately capture the context in which decisions are made (in ME). In addition, their poor interoperability with CASE and method engineering environments constrains their usefulness in supporting method rationale, as the method use decision rationale captured in these environments is mostly lost when it is separate from the contextual design knowledge stored in ME environments.

Within CASE tools, some mechanisms to capture design rationale both as annotations and as argumentations have been developed (e.g. Bigelow, 1988; Cybulski and Reed, 1992; JinXiang and Griggs, 1994; Oinas-Kukkonen, 1996; Pohl et al., 1994; Ramesh and Dhar, 1992). These tools focus on capturing and using the rationale behind the creation of artifacts during the design process. None of these tools, however, supports method rationale simultaneously during method engineering and method use. Consequently, these tools provide limited support to access knowledge of method use at the modeling level from within method modeling tools, and vice versa. Yet, as noted, this

[10] This happens in some specific cases like pilot tests, joint development projects with method developers, or in action learning (Checkland 1981, Mathiassen et al 2000)

is critical for evolutionary method engineering, which enables designers to engage in double- loop learning. To this end, one goal of this essay is to demonstrate that such an environment can be built and used to maintain all method knowledge within one unified CASE and Computer Aided Method Engineering (CAME) tool.

Evolutionary Method Engineering and A Proposed Method Rationale Architecture

Evolutionary method engineering lifecycle

The evolutionary process of developing and refining method definitions (evolutionary ME) is outlined in the data flow diagram in Figure 2. According to the model, evolutionary ME steps can be divided into setting up ME goals, constructing and adapting a method for a given situation, gathering use experience, analyzing method use, and further refining the method. This process should not be regarded as a "waterfall" like sequence but as an outline of a set of iterative activities in which gradual method improvements take place based on different method stakeholders' experience (cf. Checkland, 1981). During method selection developers make, high-level method decisions by choosing an appropriate method "frame" for each IS-envisioned development situation based on scenarios and domain analysis (Harmsen, 1997). In the next phase, method construction, a formal meta-model is specified to render available method knowledge explicit. Formal meta-models in this view provide a mechanism to collect and organize development experience and make it explicit and analyzable. Method stakeholders' comments, observations, and change requests can now be related to the types and constraints of the proposed formal meta-model. Formal meta-models also allow identification of those parts of the method, which may be in need of further analysis (e.g. assumptions, nature of the entities being analyzed).

Alternative method refinement strategies can be pursued and their outcomes compared by using meta-modeling constructs and development scenarios. When a formal meta-model is stabilized, it is adapted into a tool environment (step 3), or at least formalized into a method handbook (by-passing step 3).

Process steps 1 - 3 in Figure 2 have been the primary focus of meta-modeling and ME research in the past two decades. Most of this literature has ignored continued method refinement processes during method use and how it is driven by use experience (steps 4-6). This means that method engineers need to probe the ISD environment continuously, not just during the initial method construction, and use this knowledge to improve specific methods. Previous research has also ignored the new challenge of managing such a dynamic method portfolio in fast changing environments as discussed in section 2. As a consequence, the literature has not recognized the need for method rationale and how it relates to the desired functionality of both a CAME and a Computer Aided System Engineering (CASE) tool. Based on the concept of evolutionary ME, we claim that the collection of experiences and the introduction of incremental changes to methods are critical for supporting experience-based learning. This will lead to improved acceptance and effectiveness of methods and a better managed process for maintaining

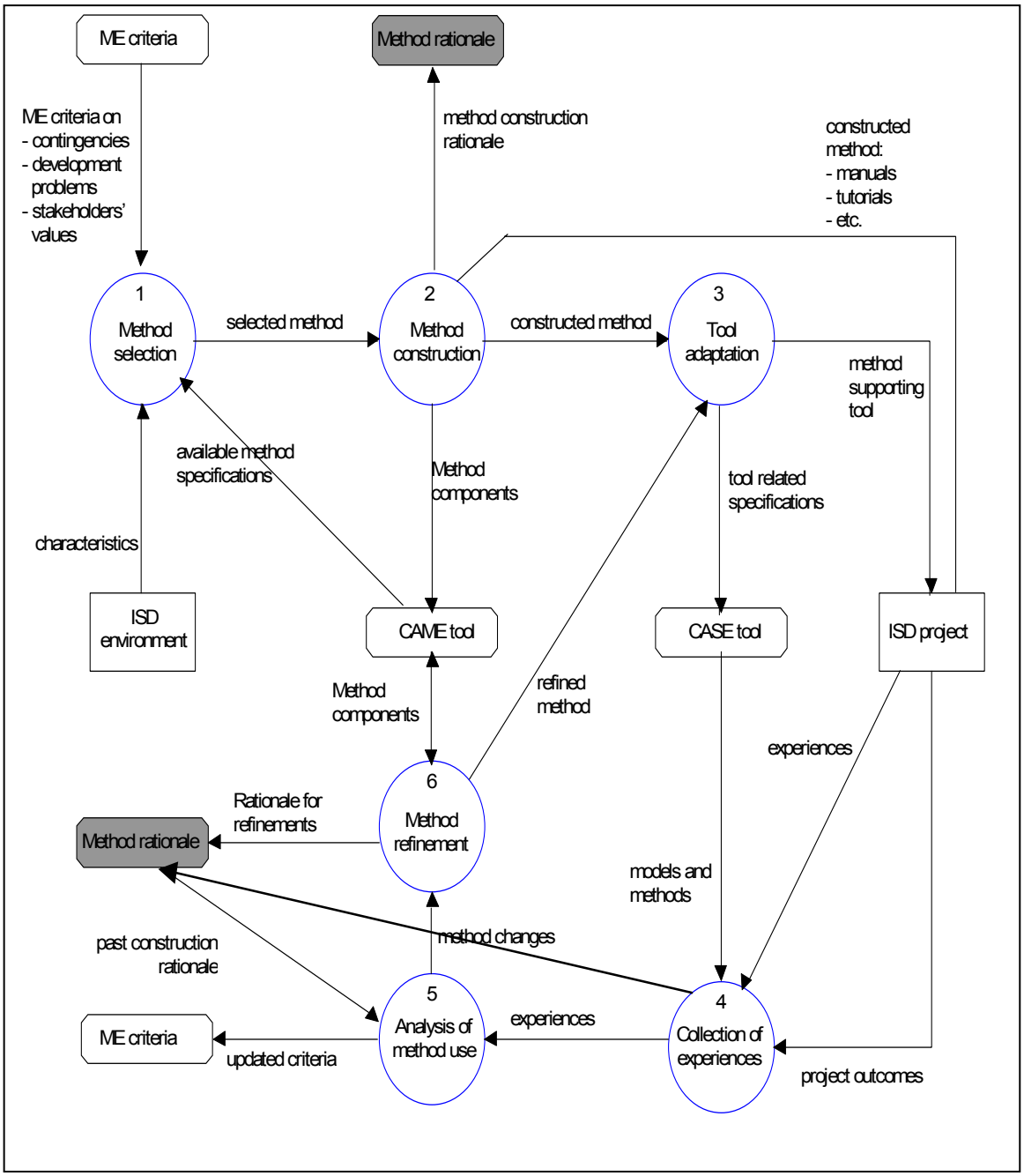

Figure 2 A Data Flow Diagram Specifying the Incremental Method Engineering Process

the method portfolio. Relating method use experience to meta-models and method construction decisions also enhances organizational learning about methods and development situations. By organizing models and meta-models into explicit and well-defined relationships (CAME/CASE relationship in Figure 1), garnering development experience, relating it to formal models, and interviewing stakeholders will all increase

the availability, and therefore the relevancy, of method use experience.[11] Because of the complexity and scope of this task, we also believe that maintaining method rationale should be regarded as an integral part of ME methods and supported by appropriate tools. This necessitates integration of both CAME and CASE[12] functionality with method rationale capability.

Kaipala (1997) was the first to suggest a set of such tools, using the IBIS model (Conklin and Begeman, 1988) for recording design decisions and collecting method use experience. We expand his model by recognizing that, in addition to the model-based ME deliverables, other improvements or refinements of the methods need to be captured in a method rationale as well. These deal with collecting information about ISD processes that change or improve the problem situation. Typically, a request for method change only becomes apparent through designers' observations of the limitations of the current method in use. Unfortunately, current CAME tools do not adequately support recording use situations, and it is largely done in an ad-hoc fashion through annotations and written memos that accompany formal meta-models. The availability of integrated tools to comment on method use and support designers' reflecting-in-action would greatly enhance the method development process, and also contribute to easier collection of change requests.

In evolutionary ME, experience-based meta-model refinements can be operationalized as method construction decisions "on-the-fly"[13], which necessitate simultaneous and in many cases instant modifications in the accompanying CASE tools and method documentation.[14] This change process is goal driven, as method changes are carried out to better satisfy some stated (or un-stated) ME criteria (see Figure 2). This validation may: 1) confirm or reject the currently applied criteria in the method construction, or 2) add totally new criteria.[15] Paradoxically, blueprint ME approaches, which specifically have aimed to follow specific criteria for ME, have neither discussed how to validate resulting methods against such criteria, nor analyzed how information about methods' situational applicability could be used in tailoring them to satisfy stated goals (e.g. Brinkkemper et al., 1995; Harmsen et al., 1994b).

---

[11] The approach to collect data and comments in an incremental ME process has similarities with ideas proposed by Fitzgerald and others when they talk about improving method use (Fitzgerald, 1991, Oinas-Kukkonen, 1997, Schipper and Joosten, 1996)

[12] These two environments are normally separate functionally and relate by simple data pipes so that a CAME tool compiles a specification used in a CASE tool. The can also be totally integrated which offers new way of relating meta-models and system models. We will discuss this more in next sub-section.

[13] This is in fact a (meta) level enhancement process in relation to method environments.

[14] As noted above, this process many times leads also to the new ME knowledge for future ME efforts.

[15] This is an example of ME-based learning too.

An Architecture for Method rationale in evolutionary method engineering

Method rationale is located at two levels depending on the type of users and the scope of method changes implied (Jarke et al., 1994; Oinas-Kukkonen, 1996). These we call the *method construction rationale,* and the *method use rationale,* depending on whether the primary users and/or producers of the method rationale components are system developers or method engineers, respectively. These two parts of method rationale impose and assume different types of type-instance relationships within the CAME and the CASE environments, and also convey different types of semantics for these relationships.

We present a general architecture for method rationale, its modeling types, and the semantics of embedded relationships in Figure 3. For method engineers, method rationale offers a type system and associated semantics that help explain why certain meta-model components (objects or constraints) are included in the constructed meta-model. Method rationale during method construction relates explanations both to the meta-model as a whole and to its specific constructs. Method construction rationale helps understand the reasons and effects of method modifications: what capabilities are gained or lost when a specific method element is added or removed. It also helps trace discussions of possible new method components that have been discarded so far.

Designers (method users) understand method rationale differently through models, which we called *method use rationale*. From their perspective, a method rationale contains a set of models that explains why certain types or constraints of the method exist, why and how they are used, and what are their specific strengths and weaknesses. In low maturity software organizations (say, at CMM level 1), all such knowledge is tacit/implicit, and shared unevenly among developers who have garnered it by trial-and-error learning. At higher levels of maturity, more and more such knowledge is made explicit, standardized, and thereby shared.[16] If all ISD knowledge could be specified at a suitable level of formality and generality so that it could be taught as a step-wise formalized process, writing up one generic meta-model and the associated method use rationale would offer a one-time, one-size fits all solution. Unfortunately, as noted above, process models and method use are far from uniform across different use contexts. Therefore, the need to explicitly collect a method use rationale across different use contexts and populations is of utmost importance. Whilst this reduces the "subjective" bias in method use and makes method use decisions more explicit for different contexts, it simultaneously allows users to directly relate their situated method experience to formalized method knowledge. This is important, as individuals have different goals and work styles. Therefore, they can have contradictory opinions of the need to use the method in a specific way. Method use rationale also gives inexperienced developers a way to learn about best practices related to a specific method.

The juxtaposition of the situational knowledge with formalized meta-models increases

16 This covers mostly however process specifications and process attributes and process and tool interactions. CMM is less specific about meta-models that deal with representations of object systems.

the applicability of meta-models for different contexts and domains.[17] This combination of situated and generic method knowledge helps erect methodical "pattern" repositories from which designers can seek and match method chunks or components (Harmsen et al., 1994a) for a given situation (Hidding et al., 1993). Finally, part of the method rationale is design rationale, which records the choice of specific design solutions created while working within the confines of a selected method. For example, the design rationale helps track good or bad outcomes of using the method or how it was used to arrive at a specific solution. A further discussion of how design rationale relates to evolutionary method engineering is however outside the scope of this study.[18]

Example of Method Rationale

In this section we describe a meta-model-based tool that can support method rationale capture and use. The model draws upon models of argumentation and helps represent the method rationale of different method design choices. The tool is implemented in the MetaEdit+ CAME tool and was mainly derived from the earlier REMAP model (Ramesh and Dhar, 1992) that was developed for capturing design decisions.

Method engineering environment

MetaEdit+ (Kelly and Smolander, 1996) is a customizable CASE environment that simultaneously supports both CASE and metaCASE functions for multiple users within a completely integrated environment. A method can be developed and simultaneously tested in method engineers' workstations much the same way as described in Hedin (1992). MetaEdit+ also supports building and integration of multiple methods (their meta-models) and offers multiple editors for diagrams, matrices, and tables and a number of analysis and reporting tools for checking model consistency and forward engineering (code generation). MetaEdit+'s architecture forms a "chain" of client-server environments. The server side contains a centralized meta-engine that uses object repository services (as a client) and acts as a server for modeling tools (diagramming, matrix, etc.). The object repository is implemented as an object database running on a central server. All editors (clients) and other tools communicate only through shared data and state information on the server that are maintained by the meta-engine. Due to this level of integration MetaEdit+ offers a high level of interoperability between its tools and "pluggable" tools. For example Kelly (1994) shows how simple it is to extend MetaEdit+ with a new tool-a matrix editor-while the information model, repository engine, and database do not change at all during the addition.

---

[17] In architecture this naturally takes longer periods of time and different architects and schools can have quite different ideas what the actual methods need to be.

[18] A detailed treatment of design rationale can be found in (Lee, 1993, Moran and Carroll, 1996, Ramesh and Dhar, 1992, Ramesh and Sengupta, 1995)

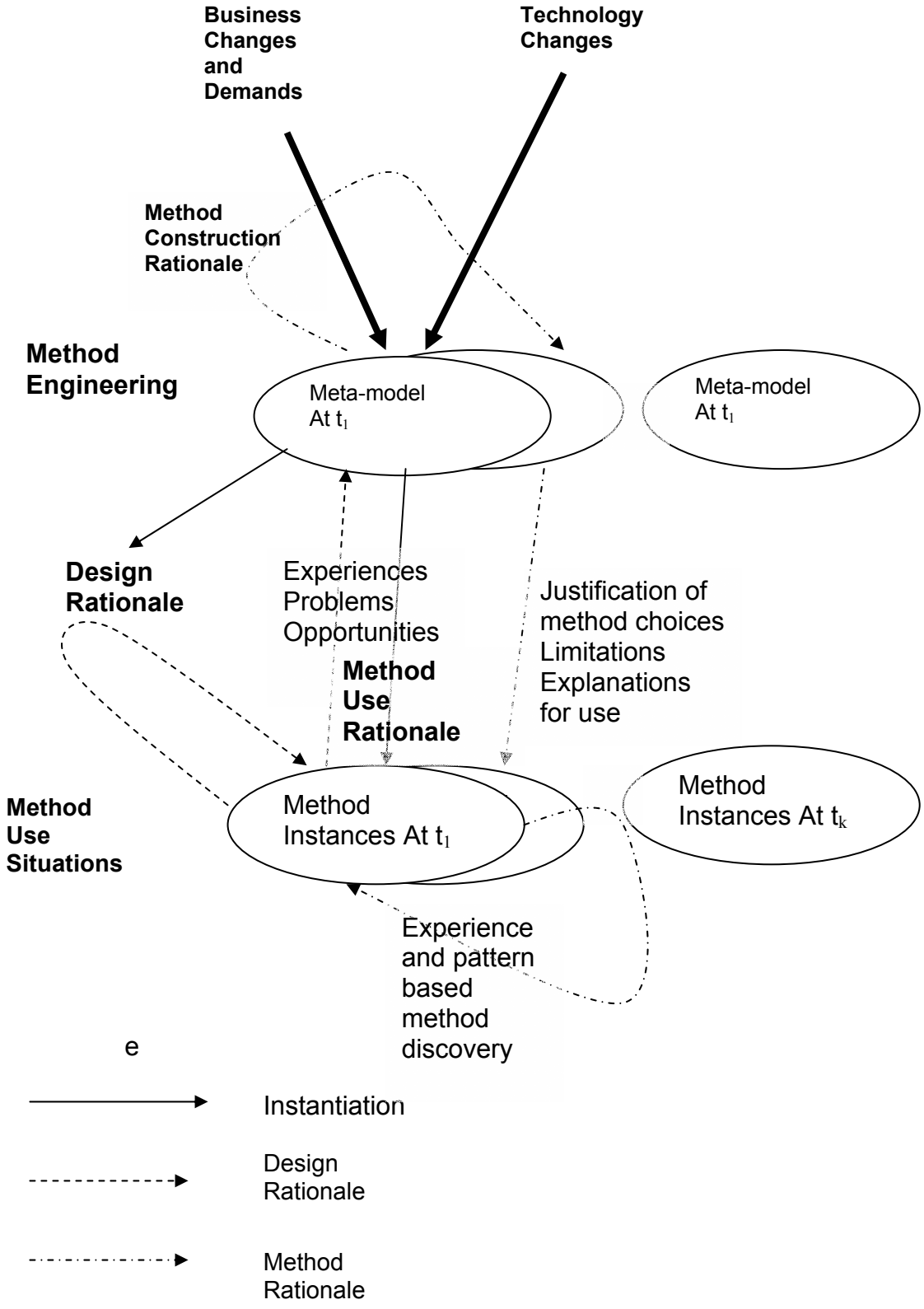

Figure 3. Method rationale and evolution

All information in MetaEdit+ is stored in the same object repository, including methods specification (meta-models), and instances of methods in the form of diagrams, matrices, or specific design objects, properties, and even font selections. Hence, modification of system designs (or methods) in one MetaEdit+ client is automatically propagated to other clients. A design repository is composed of one or more projects, each of which contains a set of graphs that describe a particular system or application, and possibly some meta-models tailored for that application. The repository definition (i.e. the set of meta-models for each project) forms the centralized representation from which modifications to both designs and methods are derived for that project, as MetaEdit+ ensures that data in repository remains consistent at all times with the defined repository schema. Because MetaEdit+ is fully object-oriented, it can organize and flexibly reuse any component in the environment from any project.

The core conceptual types of a method in MetaEdit+ are defined in the same repository, but on a higher level, and can thus be modified by the method developers and users on the fly depending on access constraints. To support visualization of designs, MetaEdit+ offers easy ways to define how each component of a method is visually represented. MetaEdit+ also offers sub-views that help customize presentations of the components and change styles of interaction (for example, domain experts can just look at the high-level descriptions of attributes, while designers can see the implementation details as well).

As MetaEdit+ allows incremental specification of methods, it also allows for incomplete meta-models, thus allowing users to model systems by using only partial method specifications. Hence, method engineers can change components of a method even while system developers are working with older versions of the method. The data continuity, (i.e. that specification data that remains usable even after method schema changes), is confirmed by a number of checks and limitations that MetaEdit+ poses to the method evolution possibilities. The idea is that the user can always be guaranteed data continuity while working with partial methods.

Method construction rationale

For this study, we defined a simplified version of the REMAP model using MetaEdit+'s method-specification language – GOPRR (Kelly et al., 1996). REMAP extended the Issue Based Information Systems (IBIS) framework for modeling argumentation processes (Conklin and Begeman, 1988). In REMAP, issues represent questions or concerns of interest. Alternatives or positions represent different ways of responding to an issue. Arguments that either support or oppose these alternatives need also to be identified. In addition, the model explicitly captures any assumption behind the above primitives. REMAP demands that each user represents the context in which the design decision was made. These decisions select one or more alternatives that can resolve the issue at hand. The REMAP model has been successfully used in a wide variety of complex organizational problem-solving situations and has been incorporated into several CASE tools (Ramesh and Jarke, 2001). Its formal meta-model helps in building tool support for design rationale. As its meta-model can be defined using the same formalism that is used to model the design methods, the linking of method components to the design rationale is normally straightforward.

In this section we build a simple set of examples of method modification decisions around the evolution of UML. We offer the examples to illustrate how method rationale, when modeled as a REMAP process within MetaEdit+, can be helpful in supporting UML deployment evolution in an organization. Though the examples are relatively small and partial, we feel that they demonstrate the applicability of recording method rationale and how to better support it with a CAME/CASE tool environment. Even in these small examples, there is a good chance that the organization will lose its former method experience, and therefore be unable to fully exploit the new possibilities, when the rationale is not recorded.

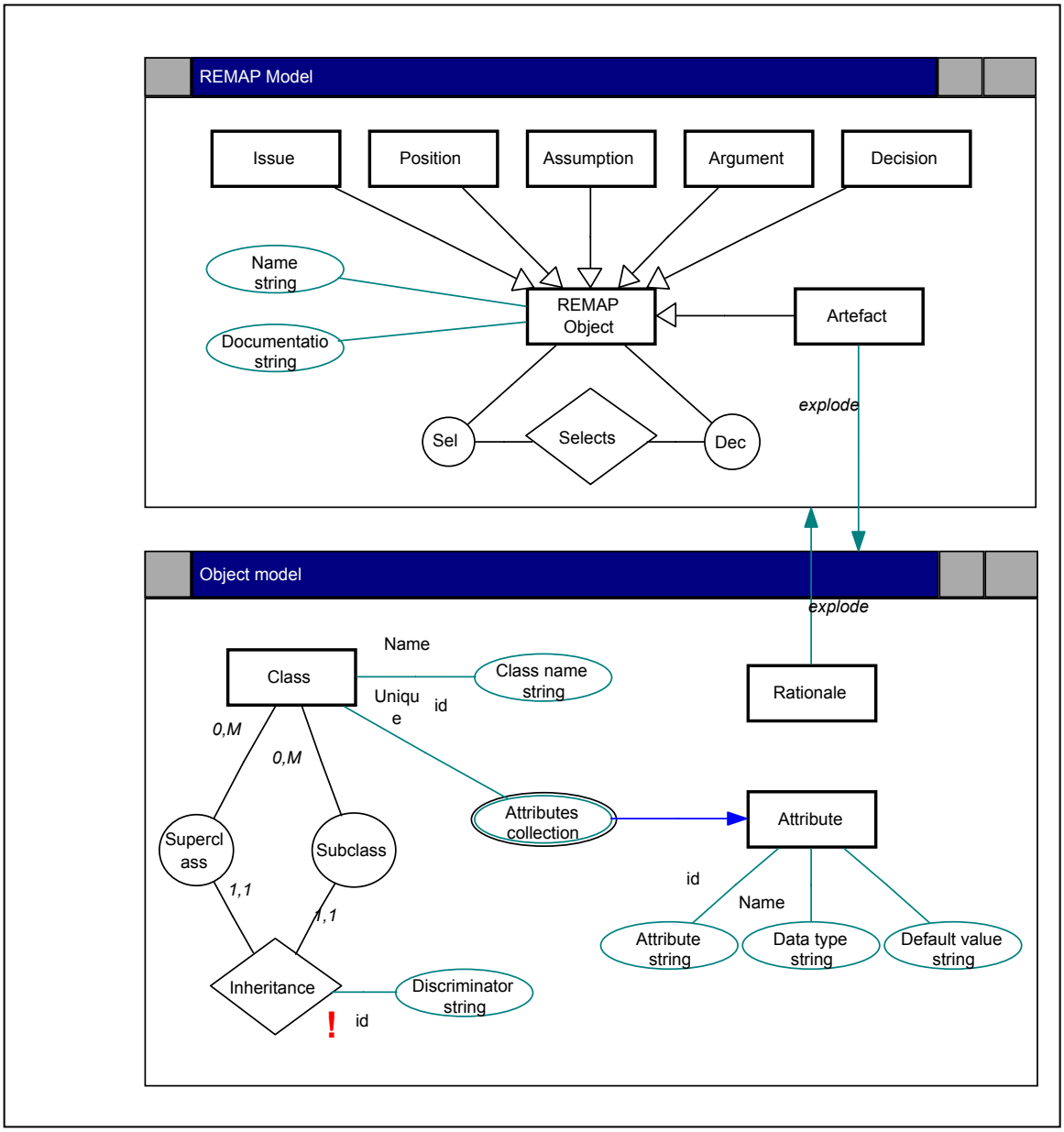

Figure 4. REMAP meta-model connecting to part of a UML Class Diagram meta model (simplified)

We assume that most parts of the UML method have been modeled previously by using the MetaEdit+ tool and are in use. Figure 4 shows a part of the current UML Class Diagram (UML 1.2 notation) specification using the GOPRR meta-modeling language (Kelly et al., 1996). The model is very similar to role based entity modeling and should be self-descriptive. Assume that the organization is now considering a move to a new implementation environment that uses Java programming language and web development tools for implementing web services. When hearing of the plans, designers immediately raise the issue that the current version of UML (1.2) does not support code generation for Java very well. At the same time, they know that a new 1.5 version of UML has been proposed as a standard, which would support better code generation for Java. Furthermore, they know that (not yet standardized) UML 2.0 will offer improved support for local variants of methods, called profiles. At the same time, UML's constraint checking mechanisms-called object constraint language (OCL) are undergoing standardization at OMG (2003).[19] These changes proposed for UML will together offer better support for architectural design of design components, as several inconsistencies in the earlier versions of the method will have been rectified. A key issue for designers is also to better support model exchanges between tools using XML-based data interchange formats.[20] However, there will be a major discontinuity and re-learning effort when the new method version is put into use.

Figure 5 shows how this situation would be reflected in the method construction rationale. The exclamation mark in the UML meta-model that deals with the inheritance model connects the current version of the meta-model to a decision whether to stick with the single inheritance for the time being, as there are also other ways to model the inheritance. The associated part of the method rationale for the meta-model is shown in the upper part of the figure. The REMAP model (Ramesh and Dhar, 1992) represents the associated method rationale behind the use of single inheritance as a constellation of issues, alternatives, supporting and objecting arguments for each alternative, and specific assumptions related to the decision about inheritance. Table 1 provides the legend for REMAP primitives shown in Figure 5. After a careful consideration of the pros and cons of using single or multiple inheritance, developers returned to the original decision to use single inheritance based on the assumption that the organization will continue to use Smalltalk programming language that does not support multiple inheritance. We show outcomes of this process n Figure 6. This figure shows an instantiation of the REMAP meta-model represented in Figure 5. This example also highlights how external factors can force or constrain changes to methods. In this case, it is the technical possibility of using multiple inheritance by switching to another programming language that may make invalid the original decision that assumed the use of Smalltalk.

---

[19] For a fuller treatment of the evolution of UML and its standardization odyssey examine (Kobryn, 1999).

[20] We will elaborate the use of method rationale by showing discussions related to these issues.

Table 1. Symbols used in Method Rationale models in Figures 5, 6 and 7		
Shape	**Shade**	**Method Rationale Component**
Oval	White	Issue
Diamond	White	Alternative
Circle	White	Argument
Rectangle	White	Assumption
Rectangle	Gray	Decision (Draft)
Rectangle	Green	Decision (Frozen)
Rectangle with border	White	Artefact (e.g., method component)

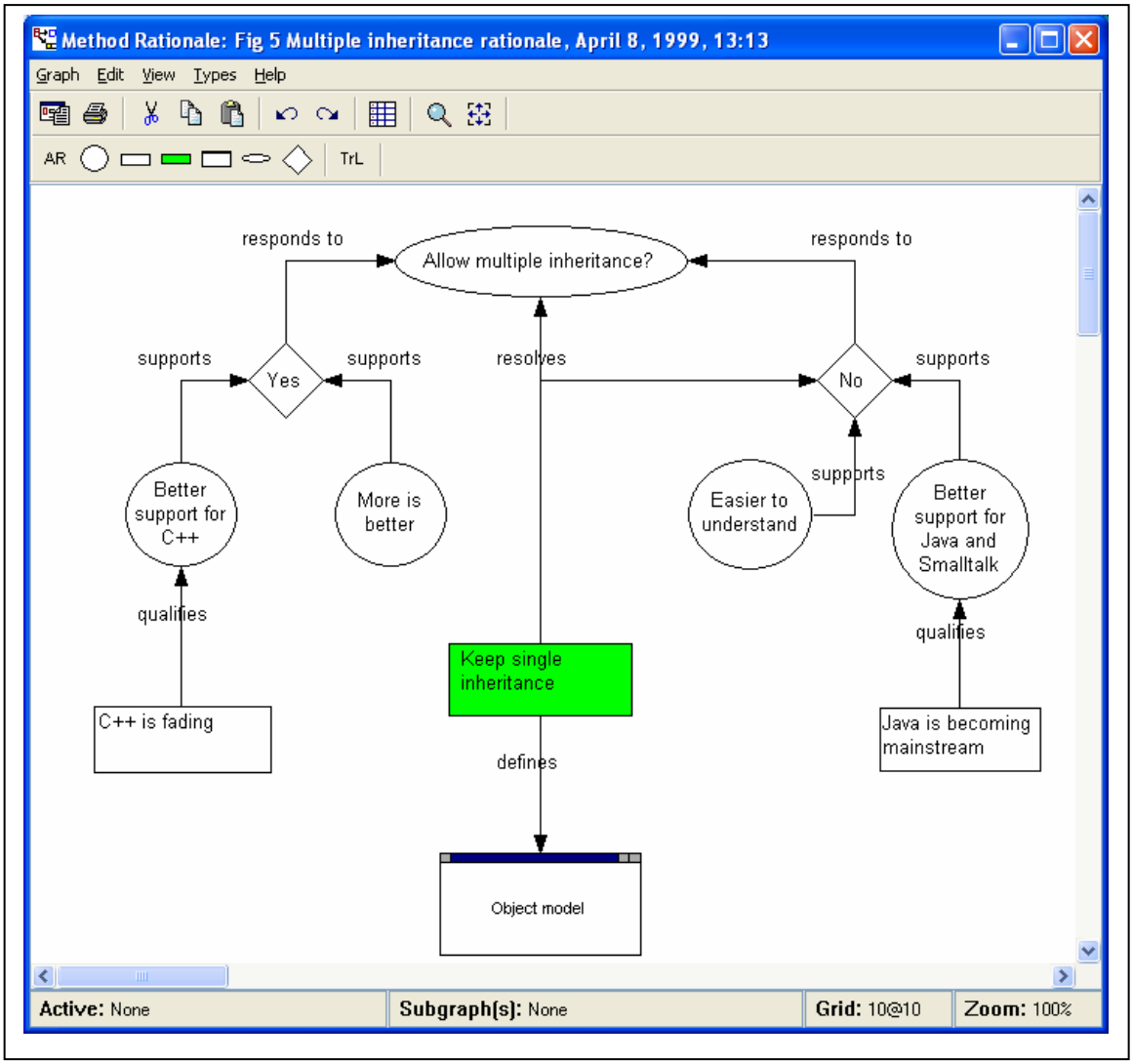

Figure 5. Rationale for a single inheritance

Method Use Rationale

The original goal of UML was to standardize object-oriented modeling methods. Yet, recently its developers have recognized the need to locally modify these methods, in particular to better serve different target domains and development environments. This is especially relevant for UML since it seeks to provide a design-oriented language that provides a higher level of abstraction over implementation detail presented in programs. To allow for specialization for different implementation platforms, UML 1.5 defines two new mechanisms: extensions and profiles. The extensions allow specializing or extending certain types, such as classes, by sub-typing them. Profiles allow for the development of domain-or organization-specific extensions of the method. Some of the more advanced developers would want to change the current UML standard so that it would also allow for the use of profiles during design time. These profiles could be used to formalize the use of special conventions and processes within each adopting organization. Figure 6 below shows the ongoing rationale for this.

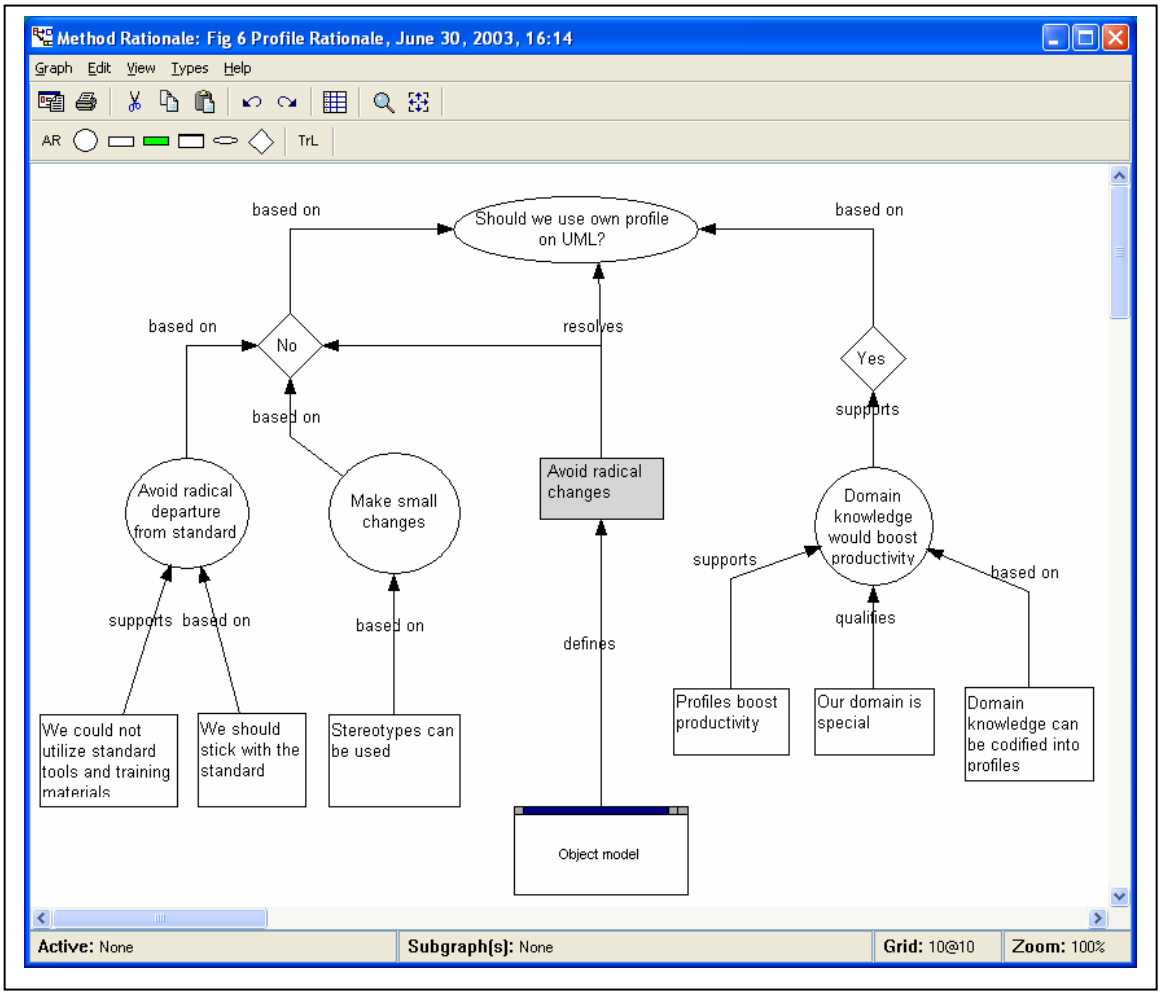

Figure 6. Rationale for own profile

The issue is unresolved at this point, and for the time being, the old method has been kept as it is. However, if a new construct- profile- were to be developed as part of the official UML meta-model, this would entail significant changes to the current meta-model. Arguably, it would start a new method development cycle. This is notified by several arguments in the lower left corner of the model.

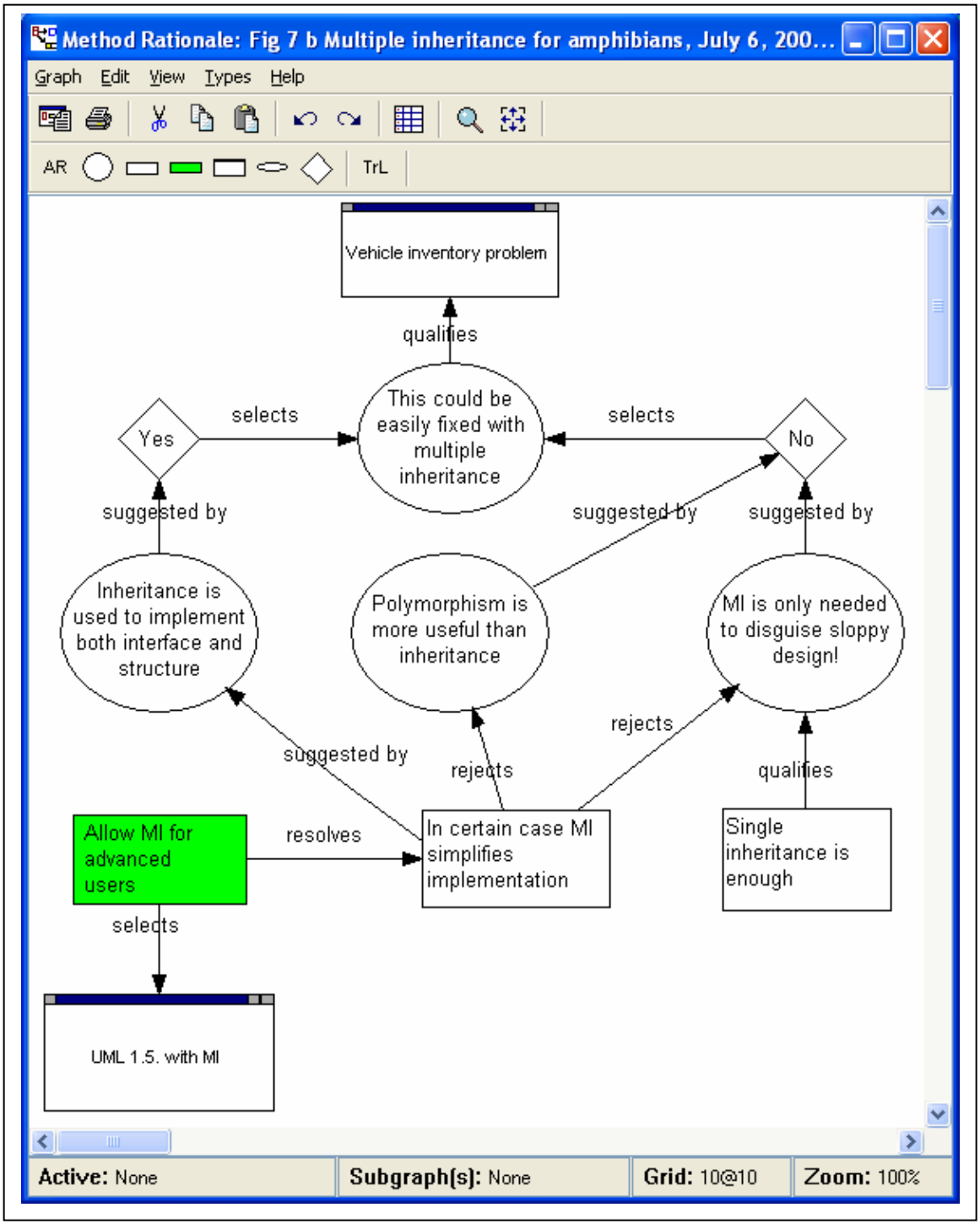

Figure 7. Multiple inheritance and rationale for using it

Now, consider a situation wherein the constructed UML version has been used in the organization with single inheritance. The developers who are familiar with multiple inheritance begin to question the original decision and would like to add support for multiple inheritance into the method. The resulting discussion is depicted in Figure 7. Here, the designers comment on a design fragment, stating how easy it would be to add more flexibility to the system if the method allowed for multiple inheritance. Thereafter, a heated argument is under way about the pros and cons of multiple inheritance. The designers finally resolve the issue by allowing for two different variants of the method (shown as part of the method rationale) that the choice of use of either single or multiple inheritance has to be made on a project-by-project basis, and the decisions and rationale for the choice have to be recorded in the project method specification.

Discussion

Method rationale and system development

The examples in the previous section offer a small glimpse of how both method construction and use rationale are built and how method use leads into a reflection about desirable method properties. If such experience receives enough attention, it can raise the need to change the method, and thus change the method construction rationale. These examples also illustrate how feedback mechanisms can be built into the CAME/CASE tools supporting rationale-based learning and method evolution.

These small examples reveal a significant gap between currently proposed ME criteria and their linkages to produced meta-models: none of them adequately relate ME requirements to individual types or constraints of a method. Some of the ME approaches support linking information about method use situations and contingencies to meta-models based on predefined coarse schemata (Harmsen et al., 1994b; Kaipala, 1997). These approaches fail, however, to explain how such detailed descriptions are obtained, nor do they relate to meta-models or their parts, which would offer enough detail to understand method change.

We deliberately chose the studied examples to illustrate what could happen when the method rationale is not kept. In increasingly complex development situations it can become impossible to understand and manage the method evolution locally when method changes are frequent. This situation is likely to become worse if and when local method variants emerge, as is now being suggested through UML extension mechanisms. One area where the evolution has already become critical is in managing knowledge assets that relate to the evolution of complex and embedded software-intensive systems, which often draw upon domain-specific modeling languages (e.g. telecommunications, web development, etc.). There is a growing number of studies that raise the issue of how to address the evolution of domain-specific methods and their variants (Kelly and Tolvanen, 2000; Weiss and Lai, 1999). To address such concerns, Microsoft, for example, is currently developing a meta development tool called WhiteHorse (Randell and Lhotka, 2004). Microsoft expects this tool to offer improved support for the use of UML class diagrams that align better with the evolving .Net framework (Randell and Lhotka, 2004). In most of these approaches, the method evolution is still typically viewed as a natural and uncontrolled process, or even drift. Our claim is that organizations can benefit more by making their whole method rationale

explicit, and thereby carefully managing evolutionary method engineering and associated method portfolios.

Implications of Method Rationale for Came/Case Support

In general, method rationale can be represented in a variety of ways, from *mathematically* formal representations (e.g., transformations that derive one method instance from another) to very informal representations (e.g., design notebooks in natural language) (Lee, 1993). Formal and informal representations of method rationale complement each other due to their respective strengths and weaknesses. Informal method rationale is easy to acquire, whereas formal representations can support automated reasoning. As observed in recent studies of design rationale (Shum and Hammond, 1994), effective schemes for the capture and use of method rationale can benefit by combining both forms of representations. A tight integration of formal and informal aspects of method rationale knowledge not only facilitates acquisition of various types of information in a format that is most appropriate, but also helps use/reuse the captured information. The semi-formal representation used in the proposed tool sought to achieve this objective.

Comprehensive representation of knowledge about MR also requires that many method rationale components need to be formally specified and linked to informal knowledge components that are unsuitable for formal reasoning. Management of this type of method rationale knowledge requires tools for constructing, querying, and maintaining structured knowledge bases. A method rationale tool with automatic inferencing capability can help to access specific knowledge and maintain the integrity of the knowledge base. Such a knowledge base can be incrementally defined and modified, thus offering additional benefits. Additional support for aggregation, classification and generalization of knowledge components will be necessary in such environments in future.

The basic premise of our work is that method engineers and method users can be better supported in their daily tasks with MR information. Services that could be provided to support these groups include facilities for easy capture of MR knowledge, management of method evolution with and to changing situations (represented as say, changes in assumptions), replay of method construction and use histories, and dependency management between different use contexts and design contexts. In large projects, the overhead involved in the detailed representation of method rationale can be considerable. Facilities to link any ME object (say, a method fragment or a decision) to its sources (originally captured in documents, meeting minutes, e-mail exchanges, etc.) can minimize the overhead. Further, this needs to be achieved in a non-intrusive manner. As the overhead involved in capturing MR knowledge can be significant, mechanisms to support automated collection and use of this information will be necessary. For example, a tool to manage the dependencies among MR components could be used to identify appropriate method components for a given situation, or to alert the method users about the changes in the situation that necessitate the re-examination of the used method. A hypermedia interface for browsing the contents of the knowledge base is helpful for providing easier access and more user-friendly presentation. Furthermore, a concept map that visually depicts the relationships among informal knowledge components would facilitate navigation through related fragments of method

knowledge.

Our implementation illustrated in the previous section provides some of these facilities within the context of a totally integrated CAME/CASE environment. Specifically, the environment currently supports the definition of queries, several graphical and textual views, and model browsing. We believe that the modular approach chosen in the MetaEdit+ tool will allow in the future to add more complex tools for MR maintenance and use, if that would be needed..

Conclusions

The paper suggests that method rationale can act as a powerful mechanism to maintain systems development knowledge in an organization. In most cases each time a method is changed or new technology is introduced, most of the knowledge relating to the old version or the need for a change is lost. Yet, when method rationale is maintained, it can provide guidance for future decisions and aid method users to become better familiar with the method. Regular method users can also benefit from this kind of method knowledge, because it makes the peculiarities and limitations of the currently used method more understandable. If method rationale was routinely recorded and if it could be used while using the method, it could aid in a dialectical exchange of ideas for modifying the method and thus improving the ISD process. This should be useful for organizations that aim at continuous development of their ISD processes and which want to learn from their experience (Lyytinen and Robey, 1999).

Our framework for method rationale improves in several ways the state of the art in computer support for systems development. First, it is a first comprehensive treatment of the aspects of method evolution and its management within a CAME/CASE tool. Second, we have outlined several ways of using and maintaining the knowledge about a development method and its use. As the development methods form one of the main asset of software organizations, increased method knowledge should be valuable to them in maintaining their knowledge about the tools and methods in use.

There are negatives to acquiring comprehensive method rationale as well. One of them is that it increases the work that is not seen as directly productive and therefore method traces are not necessarily easily recorded. Furthermore, in the absence of proper incentives schemes designers may hesitate to expose their knowledge and expertise the larger community of practice. Successful implementation of MR in practice may require changes both in the incentives and culture of the development organizations as well as in organizational processes. Clearly there is room for empirical research on the topic and we are planning to engage ourselves in such initiatives. We are especially interested in empirical studies of domain specific method evolution and maintenance.

Acknowledgements

Thanks go to senior editor Matthias Jarke and two anonymous reviewers for their thoughtful comments. One of the author's works was supported by grants from the Office of Naval Research and Air Force Research Laboratory. We are also indebted to our colleagues Steven Kelly, Pentti Marttiin, Zheying Zhang and Janne Kaipala for their

ideas and continued support. All the remaining errors remain ours.

References

Argyris, C. and D. Schön (1978) *Organizational Learning, A theory of action perspective*: Addison-Wesley.

Avrilioni, D. and P.-Y. Cunin. (2001) “Process Model Reuse Support - The OPSIS Approach.” *10th International Software Process Workshop, 2001*.

Baskerville, R., L. Levine, J. Pries-Heje, B. Ramesh et al. (2001) "How Internet Software Companies Negotiate Quality," *IEEE Computer* (34) 5, pp. 51-57.

Baskerville, R., J. Travis, and D. Truex (1992) “Systems Without a Method: The Impact of New Technologies on Information Systems Development Projects,” in K. E. Kendall, K. Lyytinen, and J. I. DeGross (Eds.) *The Impact of Computer Supported Technologies on Information Systems Development*, Amsterdam: North-Holland, pp. 61-93.

Beck, K. and M. Fowler (2000) *Planning Extreme Programming*. New York, NY: Addison Wesley Longman.

Bigelow, J. (1988) "Hypertext and CASE," *IEEE Software* (14) 3, pp. 23-27.

Brinkkemper, S. (1990) Formalisation of Information Systems Modelling. Ph.D. Thesis, Univ. of Nijmegen.

Brinkkemper, S. (1996) "Method engineering: engineering of information systems development methods and tools," *Information & Software Technology* (38) 6, pp. 275-280.

Brinkkemper, S., F. Harmsen, and H. Oei. (1995) “Configuration of Situational Process Models: an Information Systems Engineering Perspective.” *European Workshop on Software Process Technology, 1995*, pp. 193-196.

Checkland, P. B. (1981) *Systems Thinking, Systems Practice*. New York: J. Wiley.

Cockburn, A. (2000) "Selecting a Project’s Methodology," *IEEE SOFTWARE* (July.

Conallen, J. (1999) "Modeling Web Application Architectures with UML," *Communications of the ACM* (42) 10, pp. 63-71.

Conklin, J. and M. L. Begeman (1988) "gIBIS: A Hypertext Tool for Exploratory Policy Discussion," *ACM Transactions on Office Information Systems* (6) 4, pp. 303-331.

Curtis, B., H. Krasner, and N. Iscoe (1988) "A field study of the software design process for large systems," *Communications of the ACM* (31) 11, pp. 1268-1287.

Cybulski, J. L. and K. Reed (1992) "A Hypertext-Based Software Engineering Environment," *IEEE Software* (18) 3, pp. 62-68.

Davis, G. B. (1982) "Strategies for information requirements determination," *IBM Systems Journal* (21) 1, pp. 4-30.

Demirors, O., E. Demirors, A. Tarhan, and A. Yildiz. (2000) “Tailoring ISO/IEC 12207 for instructional software development.” *The 26th Euromicro Conference, 2000* 2.

Desmond, F., A. D'Souza, and W. Cameron (1998) *Objects, Components, and Frameworks With Uml : The Catalysis Approach*: Addison Wesley Publishing Company.

Falkenberg, E. D., W. Hessa, P. Lindgreen, B. Nilsson et al. (1996) *A Framework of Information System Concepts: Summary of the FRISCO Report*. The IFIP WG 8.1 Task Group FRISCO.

Fischer, G., A. G. Lemke, R. McCall, and A. I. Morch (1991) "Making Argumentation Serve Design," *Human-Computer Interaction* (6) 3&4, pp. 393-419.

Fitzgerald, G. (1991) "Validating new information systems techniques: a retrospective analysis," in H.-E. Nissen, H. K. Klein, and R. Hirschheim (Eds.) *Information Systems Research: Contemporary Approaches and Emergent Traditions*: Elsevier Science Publishers B.V., pp. 657-672.

Floyd, C. (1986) "A comparative evaluation of systems development methods." *IFIP WG 8.1 Working Conference on Comparative Review of Information Systems Design Methodologies: Improving the Practice, Amsterdam, 1986*.

Floyd, C. (1987) "Outline of the paradigm change in software engineering," in G. Bjerknes, P. Ehn, and M. King (Eds.) *Computers and Democracy: A Scandinavian Challenge*, Brookfield Vermont: Avebury Gower.

Franckson, M. (1994) "The Euromethod Deliverable Model and its contribution to the objectives of Euromethod," in A. A. Verrijn-Stuart and T. W. Olle (Eds.) *Methods and Associated Tools for the Information Systems Life Cycle*: Elsevier, pp. 131-149.

Gigch, J. v. (1991) *Systems design and modeling and metamodeling*. New York: Plenum Press.

Ginsberg, M. P. and L. H. Quinn. (1995) *Process Tailoring and the Software Capability Maturity Model*. Software Engineering Institute CMU/SEI-94-TR-024.

Harmsen, F. (1997) Situational Method Engineering. PhD Thesis, University of Twente.

Harmsen, F., S. Brinkkemper, and H. Oei. (1994a) "A Language and Tool for the Engineering of Situational Methods for Information Systems Development." *Fourth International Conference on Information Systems Development, Kranj, Slovenia, 1994a*, pp. 206-214.

Harmsen, F., S. Brinkkemper, and H. Oei (1994b) "Situational Method Engineering for Information System Project Approaches," in A. A. Verrijn-Stuart and T. W. Olle (Eds.) *Methods and Associated Tools for the Information Systems Life Cycle (A-55)*: Elsevier Science B.V. (North-Holland), pp. 169-194.

Heberling, M., C. Maier, and T. Tensi. (2002) "Visual Modelling and Managing the Software Architecture Landscape in a large Enterprise by an Extension of the UML." *Second Domain-specific Modeling Languages Workshop, Seattle, WA, 2002*.

Hedin, G. (1992) Incremental Semantic Analysis. PhD Thesis, Lund University.

Hidding, G. J., J. K. Joseph, and G. M. Freund (1993) "Method Engineering at Andersen Consulting: Task Packages, Job Aids and Work Objects," in *2nd International Summerschool on Method Engineering and Meta Modelling conference binder*, Enschede, the Netherlands: Univ. of Twente.

Highsmith, J. (1999) *Beyond RAD: Reducing cycle time through innovative management*. Arlington, MA: Cutter Information Corp.

Hofstede, A. H. M. t. and T. F. Verhoef (1996) "Meta-CASE: Is the Game Worth the Candle?," *Information Systems Journal*) 6, pp. 41-68.

Holdsworth, J. (1999) *Software Process Design: out of the tar pit*. London, U.K.: McGraw-Hill International (UK) Limited.

Hollenbach, C. and W. Frakes. (1996) "Software Prosess Reuse in an Industrial Setting." *the 4th International Conference on Software Reuse, 1996*.

Hughes, J. and E. Reviron (1996) "Selection and evaluation of information system development methodologies: The gap between the theory and practice," in N. Jayaratna and B. Fitzgerald (Eds.) *Lessons learned from the use of methodologies*, pp. 309-319.

Iansiti, M. and A. MacCormack (1997) "Developing Products on Internet Time," *Harvard Business Review*) October, pp. 108-117.

IEEE/EIA (1998a) "Industry implementation of International Standard ISO/IEC 12207.0: 1995. (ISO/IEC 12207 standard for information technology - software life cycle processes - implementation considerations," *IEEE/EIA*.

IEEE/EIA (1998b) "Industry implementation of International Standard ISO/IEC 12207.1: 1995. (ISO/IEC 12207 standard for information technology - software life cycle processes - implementation considerations," *IEEE/EIA*.

IEEE/EIA (1998c) "Industry implementation of International Standard ISO/IEC 12207.2: 1995. (ISO/IEC 12207 standard for information technology - software life cycle processes - implementation considerations," *IEEE/EIA*.

Jarke, M., K. Pohl, C. Rolland, and J.-R. Schmitt (1994) “Experience-Based Method Evaluation and Improvement: A Process Modelling Approach,” in A. A. Verrijn-Stuart and T. W. Olle (Eds.) *Methods and Associated Tools for the Information Systems Life Cycle (A-55)*: Elsevier Science B.V. (North-Holland), pp. 1-27.

Jarke, M., K. Pohl, K. Weidenhaupt, K. Lyytinen et al. (1998) “Meta Modeling: A Formal Basis for Interoperability and Adaptability,” in B. Krämer and M. Papazoglou (Eds.) *Information Systems Interoperability*: John Wiley Research Science Press, pp. 229-263.

JinXiang and Griggs. (1994) “A Tool for Hypertext-based Systems Analysis and Dynamic Evaluation.” *The 27th Annual Hawaii International Conference on system Sciences, Hawaii, 1994*.

Kaipala, J. (1997) “Augmenting CASE Tools with Hypertext: Desired Functionality and Implementation Issues.” *Advanced Information Systems Engineering, Barcelona, Spain, 1997*, pp. 217-230 1250.

Karlsson, F. (2002) “Bridging the Gap - between Method for Method Configuration and Situational Method Engineering.” *The Second Annual Knowledge Foundation Conference for the Promotion of Research in IT, Stockholm, Sweden, 2002*.

Karlsson, F., P. J. Agerfalk, and A. Hjalmarsson. (2001) “Method configuration with development tracks and generic project types.” *The 6th CAiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in System Analysis and Design, Switzerland, 2001*.

Kelly, S. (1994) "A Matrix Editor for a MetaCASE Environment," *Information & Software Technology* (36) 6, pp. 361-371.

Kelly, S., K. Lyytinen, and M. Rossi (1996) “MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment,” in P. Constapoulos, J. Mylopoulos, and Y. Vassiliou (Eds.) *Advanced Information Systems Engineering, proceedings of the 8th International Conference CAISE'96*, Berlin: Springer-Verlag, pp. 1-21.

Kelly, S. and K. Smolander (1996) "Evolution and Issues in MetaCASE," *Information & Software Technology* (38) 4, pp. 261-266.

Kelly, S. and J.-P. Tolvanen. (2000) “Visual domain-specific modeling: Benefits and experiences of using metaCASE tools.” *International workshop on Model Engineering, 2000*.

Kobryn, C. (1999) "UML 2001: A Standardization Odyssey," *Communications of the ACM* (42) 10, pp. 29 - 37.

Kraiem, N., I. Bourguiba, and S. Selmi. (2000) “Situational Method For Information System Project.” *International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science, and e-Medicine on the Internet, 2000*.

Kruchten, P. (2000) *The Rational Unified Process: An Introduction*, 2nd edition. Reading, MA: Addison-Wesley.

Kumar, K. and R. J. Welke (1992) “Methodology Engineering: A Proposal for Situation Specific Methodology Construction,” in W. W. Kottermann and J. A. Senn (Eds.) *Challenges and Strategies for Research in Systems Development*, Washington: John Wiley & Sons, pp. 257-269.

Lee, J. (1991) “A Qualitative Decision Management System,” in P. Winston and S. Shellard (Eds.) *Artificial intelligence at MIT:Expanding Frontiers*: MIT Press: Cambridge, Massachusetts.

Lee, J. (1993) “Design Rationale Capture and Use,” in *AI Magazine*, vol. 14, pp. 24-26.

Lindgreen, P. (1990) *A Framework of Information Systems Concepts, Interim report of the IFIP WG8.1 Task Group FRISCO*.

Lyytinen, K. (1987) "Different Perspectives on Information Systems: Problems and Solutions," *ACM Computing Surveys* (19) 1, pp. 5-46.

Lyytinen, K. and D. Robey (1999) "Learning Failure in Information System Development," *Information Systems Journal* (9) 2, pp. 85-101.

Lyytinen, K., K. Smolander, and V.-P. Tahvanainen. (1989) “Modelling CASE Environments in Systems Development.” *First Nordic Conference on Advanced Systems, Stockholm, 1989*.

Machado, C. F., L. C. d. Oliveira, and R. A. Fernandes. (1999) “Experience Report - Restructure of Processes based on ISO/IEC 12207 and SW-CMM in CELEPAR.” *Fourth IEEE International Symposium and Forum on Software Engineering Standards, 1999*.

MacLean, A., R. Young, V. Bellotti, and T. Moran (1991) "Questions, Options, and Criteria: Elements of A Design Rationale for User Interfaces," *Journal of Human Computer Interaction* (6pp. 201-250.

Mathiassen, L., A. Munk-Madsen, P. A. Nielsen, and J. Stage (2000) *Object Oriented Analysis & Design*, 1 edition. Aalborg: Marko Publishers.

Moran, T. P. and J. M. Carroll (1996) *Design Rationale: concepts, techniques and use*. New Jersey: Lawrence Erlbaum Associates.

Nonaka, I. (1994) "A dynamic theory of organizational knowledge creation," *Organization Science* (5) 1, pp. 14-37.

Odell, J. J. (1996) “A Primer to Method Engineering,” in S. Brinkkemper, K. Lyytinen, and R. J. Welke (Eds.) *Method Engineering: Principles of Method Construction and Tool Support*: Chapman & Hall, pp. 1 - 7.

Oinas-Kukkonen, H. (1996) “Method Rationale in Method Engineering and Use,” in S. Brinkkemper, K. Lyytinen, and R. Welke (Eds.) *Method Engineering, Principles of Method Construction and Support*: Chapman-Hall, pp. 87-93.

Oinas-Kukkonen, H. (1997) Improving the Functionality of Software Design Environments by Using Hypertext Technology. PhD Thesis, University of Oulu.

OMG (2003) "Unified Modeling Language," Object Management Group, http://www.uml.org/ (20.06.2003, 2003).

Orlikowski, W. (1996) "Improvising Organizational Transformation Over Time: A Situated Change Perspective," *Information Systems Research* (7) 1, pp. 63 - 92.

Orr, J. (1990) “Sharing knowledge, celebrating identity: War stories and community memory in a service culture,” in D. S. Middleton and D. Edwards (Eds.) *Remembering: Memory in Society*, London: SAGE, pp. 169-189.

Pohl, K., R. Dömges, and M. Jarke (1994) “PRO-ART: PROcess based Approach to Requirements Traceability,” in *Poster Outlines: 6th Conference on Advanced Information Systems Engineering, Utrecht, Netherlands, June 1994*.

Polo, M., M. Piattini, F. Ruiz, and C. Calero. (1999a) “MANTEMA: a Complete Rigorous Methodology for Supporting Maintenance based on The ISO/IEC 12207 Standard.” *The Third European Conference on Software Maintenance and Reengineering, 1999a*.

Polo, M., M. Piattini, F. Ruiz, and C. Calero. (1999b) “MANTEMA: a Software Maintenance Methodology Based on the ISO/IEC 12207 Standard’.” *Fourth IEEE International Symposium and Forum on Software Engineering Standards, 1999b*.

Punter, T. and K. Lemmen (1996) "The MEMA-model: towards a new approach for Method Engineering," *Information and Software Technology* (38.

Ralyte, J. and C. Rolland. (2001) “An approach for method reengineering.” *ER, 2001*.

Ramesh, B., R. Baskerville, and J. Pries-Heje (2002) "Internet Software Engineering : A different class of Processes," *Annals of Software Engineering* (14) 1-4, pp. 169-195.

Ramesh, B. and V. Dhar (1992) "Supporting Systems Development by capturing Deliberations during Requirements Engineering," *IEEE Transactions on Software Engineering* (18pp. 498-510.

Ramesh, B. and M. Jarke (2001) "Toward Reference Models for Requirements Traceability," *IEEE Transactions on Software Engineering* (27) 1, pp. 58-93.

Ramesh, B. and K.Sengupta (1995) "Multimedia in a Design Rationale Decision Support System," *Decision Support Systems* (15.

Randell, B. A. and R. Lhotka (2004) “Bridge the Gap Between Development and Operations with Whitehorse,” in *MSDN Journal*, vol. 19.

Reeves, B. and F. Shipman. (1992) “Making It Easy for Designers to Provide Design Rationale.” *AAAI’92 Workshop on Design Rationale Capture and Use, San Jose, CA, 1992*.

Rising, L. and N. S. Janoff (2000) "The Scrum software development process for small teams," *IEEE Software* (17) 4, pp. 26 -32.

Rose, G. and K. Lyytinen (2003) "The Disruptive Nature of Information Technology Innovations: The Case of Internet Computing in Systems Development Organizations," *MIS Quarterly* (27) 4, pp. 557-595.

Rosenberg, D. and K. Scott (1999) *Use Case Driven Object Modeling with UML: A Practical Approach*. Reading, MA: Addison-Wesley.

Russo, N. L. and J. L. Wynekoop (1995) “The Use and Adaptation of System Development Methodologies,” in M. Khosrowpour (Ed.) *Managing Information & Communications in a Changing Global Environment: Proceedings of the Information Resources Management Association International Conference*, Atlanta: Idea Group Publishing.

Savolainen, V., J. Geels, and J. Niemeier. (1990) *SESAM, the HECTOR Methods and Tools Database Report of ESPRIT 2082 Project HECTOR: Harmonized Concepts and Tools for Organizational Information Systems*. Fraunhofer Institute for Industrial Engineering.

Schipper, M. and S. Joosten. (1996) “A Validation Procedure for Information Systems Modeling Techniques.” *Workshop of Evaluation of Modeling Methods in Systems Analysis and Design EMMSAD'96, 1996*.

Schön, D. (1983) *The Reflectice Practitioner*. New York: Basic Books Inc.

Shum, S. and N. Hammond (1994) "Argumentation-Based Design Rationale: What use and What Cost?," *International Journal on Computer Studies* (40pp. 603-652.

Stamper, R. K. (1990) *A Semantic Analysis of Basic Concepts A contribution to FRISCO Task Group of IFIP*.

Suchman, L. (1987) *Plans and Situated Actions: The Problem of Human-Machine Communication*. Cambridge: Cambridge University Press.

Thomke, S. and D. Reinertsen (1998) "Agile product development: Managing development flexibility in uncertain environments," *California Management Review* (41) 1, pp. 8-30.

Tolvanen, J.-P. (1995) "Incremental Method Development for Business Modelling: An Action Research Case Study." *6th Workshop on the Next Generation of CASE Tools, NGCT'95, Paris, 1995*, pp. 79-98.

Tolvanen, J.-P. (1998) Incremental Method Engineering with Modeling Tools: Theoretical Principles and Empirical Evidence. Dissertation. Dissertation, University of Jyväskylä.

Tolvanen, J.-P. and S. Kelly (2000) "Benefits of MetaCASE: Nokia Mobile Phones Case Study," MetaCase Consulting plc., http://www.metacase.com/papers/MetaEdit_in_Nokia.pdf (1/7, 2004).

Tolvanen, J.-P. and M. Rossi. (1996) *Metamodeling Approach to Method Comparison: A Survey of a Set of ISD Methods*. University of Jyväskylä WP-34.

Tolvanen, J.-P., M. Rossi, and H. Liu (1996) "Method Engineering: Current Research Directions and Implications for Future Research," in S. Brinkkemper, K. Lyytinen, and R. Welke (Eds.) *Method Engineering, Principles of Method Construction and Support*: Chapman-Hall, pp. 296-317.

Weiss, D. and C. T. R. Lai (1999) *Software Product-line Engineering*: Addison Wesley Longman.

Wood-Harper, T. (1985) "Research Methods in Information Systems: Using Action Research," in E. Mumford, R. Hirschheim, G. Fitzgerald, and A. T. Wood-Harper (Eds.) *Research Methods in Information Systems*: Elsevier Science Publishers, pp. 169-191.

Zmud, R. W. (1980) "Management of Large Software Development Efforts," *MIS Quarterly* (4) 2.

About the authors

Matti Rossi is an acting Professor of Information Systems and director of the electronic business program for professionals at Helsinki School of Economics. He has worked as research fellow at Erasmus University Rotterdam and as a visiting assistant professor at Georgia State University, Atlanta. He received his Ph.D. degree in Business Administration from the University of Jyväskylä in 1998. He has been the principal investigator in several major research projects funded by the Technological Development Center of Finland and Academy of Finland. His research papers have appeared in journals such as Information and Organization, Information and Management and Information Systems, and over twenty of them have appeared in conferences such as ICIS, HICSS and CAiSE.

Balasubramaniam Ramesh is Professor of Computer Information Systems at Georgia State University. His research work has appeared in several leading conferences and journals including the IEEE Transactions on Software Engineering, Annals of Software Engineering, Communications of the ACM, IEEE Computer, IEEE Software, IEEE Internet Computing, IEEE Intelligent Systems, Annals of Operations Research and Decision Support Systems. His research focuses on supporting complex organizational processes such as requirements management and traceability, business process

management and new product development. His areas of specialization include knowledge management, data mining and e-services. His work has been funded by several grants from leading government and private industry sources such as the NSF, DARPA, ONR, ARL and Accenture and has been incorporated in several CASE tools.

Kalle Lyytinen is Iris S. Wolstein professor at Case Western Reserve University. He serves currently on the editorial boards of several leading IS journals including*, Journal of AIS journal (*Senior Editor*), Information Systems Research, Journal of Strategic Information Systems, Information&Organization, Requirements Engineering Journal,* and *Information Systems Journal* among others. He has published over 150 scientific articles and conference papers and edited or written eight books on topics related to system design, method engineering, implementation, software risk assessment, computer supported cooperative work, standardization, and ubiquitous computing. He is currently involved in research projects that look at the IT induced innovation in software development, architecture and construction industry, design and use of ubiquitous applications in health care, high level requirements model for large scale systems, and the development and adoption of broadband wireless standards and services, where his recent studies have focused on South Korea and the U.S.

Juha-Pekka Tolvanen is the CEO of MetaCase. He received his Ph.D. in 1998 from the University of Jyväskylä, Finland. His area of expertise is in engineering of software development methods for application-specific needs. In this role, Dr. Tolvanen has acted as a consultant worldwide for method development and he has published papers on software development methods in several journals and conferences.

Journal of the Association for Information Systems

ISSN: 1536-9323