JYVÄSKYLÄ STUDIES IN COMPUTER SCIENCE, ECONOMICS AND STATISTICS

43

Pentti Marttiin

Customisable Process Modelling Support and Tools for Design Environment

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 1998

JYVÄSKYLÄ STUDIES IN COMPUTER SCIENCE, ECONOMICS AND STATISTICS 43

Pentti Marttiin

Customisable Process Modelling Support and Tools for Design Environment

Esitetään Jyväskylän yliopiston yhteiskuntatieteellisen tiedekunnan suostumuksella
julkisesti tarkastettavaksi yliopiston vanhassa juhlasalissa (S212)
toukokuun 9. päivänä 1998 kello 12.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Social Sciences of the University of Jyväskylä,
in Auditorium S212, on May 9, 1998 at 12 o'clock noon.

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 1998

Customisable Process Modelling Support and Tools for Design Environment

JYVÄSKYLÄ STUDIES IN COMPUTER SCIENCE, ECONOMICS AND STATISTICS 43

Pentti Marttiin

Customisable Process Modelling Support and Tools for Design Environment

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 1998

Editors
Markku Sakkinen
Department of Computer Science, University of Jyväskylä
Kaarina Nieminen
Publishing Unit, University Library of Jyväskylä

URN:ISBN:978-951-39-9068-8
ISBN 978-951-39-9068-8 (PDF)
ISSN 0357-9921

Jyväskylän yliopisto, 2022

ISBN 951-39-0212-9
ISSN 0357-9921

Jyväskylä University Printing House, Jyväskylä
and ER-Paino Ky, Lievestuore 1998

ABSTRACT

Marttiin, Pentti
Customisable Process Modelling Support and Tools for Design Environment
Jyväskylä: University of Jyväskylä, 1998, 238 p.
(Jyväskylä Studies in Computer Science, Economics and Statistics,
ISSN 0357-9921; 43)
ISBN 951-39-0212-9
Finnish summary
Diss.

Systems engineering pursues dual goals of productivity and quality, and many organisations try to improve their design practices using methods. Computer Aided Software Engineering (CASE) and Process Centred Software Engineering (PCSE) technologies have produced many design environments to facilitate use of methods. Methods are often modified locally and adapting a design environment to support them is not straightforward. In this situation of unpredictable needs only customisable design environments (metaCASE environments) are powerful enough to provide support.

Recently, design environments have been developing from single-user taskware towards multi-user teamware and groupware. What kind of multi-user support is required in design environments is still an open issue. To answer this question we need more empirical studies tackling the feasibility of the multi-user features. On the other hand, to be able to perform a broad evaluation of these features we still need to develop new solutions for design environments. In single-user metaCASE environments the focus has been on specifying meta-data, i.e. modelling techniques and their integration. In groupware we should also tackle ways to customise processes and agents. In this study we focus on developing customisable support for process models, which guide users in their design tasks and co-ordinate multiple users' work.

This study follows a constructive research approach, and includes theory building, systems development, and observations/experimentation as research methods. As an outcome of theory building we present a multi-level model to represent various metamodelling aspects for future design environments. The developed system – a customisable process modelling environment (CPME) – provides tools for customisation of both process models and their conceptual basis (process metamodels). CPME forms an extension to the MetaEdit+ metaCASE environment, and reuses its solutions and tools in an object-oriented manner. Observations/experimentation contain eperiments with metaCASE environments, a review of process centred environments, and a test of CPME to define a process modelling language and an ISPW-6 process example.

Keywords: CASE environments, PCSE environments, metaCASE, metamodelling, method engineering, process modelling, co-ordination.

ACM Computing Review Categories:

D.2.1. Software Engineering: Requirements/Specifications:
Languages, Methodologies, Tools
D.2.2. Software Engineering: Design Tools and Techniques:
Computer-aided software engineering (CASE)
D.2.9. Software Engineering: Management:
Software process models
D.2.10. Software Engineering: Design:
Methodologies, Representation
H.5.3. Information Systems: Group and Organization Interfaces:
Asynchronous interaction

Author's Address:
Pentti Marttiin
Nokia Research Center
P.O. Box 422
FIN-00045 Nokia Group
Finland
Email: pentti.marttiin@nokia.research.com
Fax: +358 9 4376 6229

ACKNOWLEDGEMENTS

The early funding for this thesis was provided through the MetaPHOR and CAMSO projects (funded by the Academy of Finland and the Ministry of Education). More recently, this thesis was supported by the grants of Jenny and Antti Wihuri Foundation, the University of Jyväskylä and COMAS Graduate School. Since January 1998 I have been working at the Software Technology Laboratory at the Nokia Research Center.

I would like to express my gratitude to my supervisor, Professor Kalle Lyytinen. His continuous encouragement and insightful comments on research directions have been particularly valuable for my studies. I have been fortunate to work in the MetaPHOR research group, which has created an excellent atmosphere for the research without forgetting the real world (with MetaCase Consulting as an outcome). I would like to thank Kari Smolander and Veli-Pekka Tahvanainen for their guidance during my first years as a researcher. My long term colleagues Steven Kelly, Matti Rossi, and Juha-Pekka Tolvanen have given valuable comments, corrections and critisism to many parts of this study and joined in many travels abroad. In addition Hui Liu, Janne Kaipala, Minna Koskinen, Harri Oinas-Kukkonen, Simo Rossi and Tero Sillander have all influenced this study. Especially I appreciate the persistent work of Minna Koskinen, whose implementations are valuable for the study.

I am indepted to Dr. Sjaak Brinkkemper for our long-standing co-operation on method engineering research. He has reviewed many papers, advised in research directions during my stay at Twente, and also acted as an external reviewer of this study. I would like to thank another external reviewer Professor Jouni Similä for his valuable work, Dr. Frank Harmsen for our joint research discussed in this study, and also Prof. Pentti Kerola, Mauri Leppänen, and Assoc. Prof. Hannu Kangassalo, who have commented the papers included in this study. In addition, I would like to mention the Finnish doctoral program on information systems, which has organised several forums (ICIS doctoral consortium and KISS seminars at Kilpisjärvi) to present research, and to discuss with other researchers.

Finally, very special thanks are due to my parents, who have supported me throughout the whole study process, and to Anna-Maija, whose encouragement has continued even though the practicalities of this study have forced us to live apart for long periods.

CONTENTS

CHAPTER 1

INTRODUCTION AND OVERVIEW

"A state without the means of some change
is without the means of its conservation"
- Edmund Burke (1729-97): Reflections on the revolution in France

1 INTRODUCTION

1.1 Motivation

One of the primary goals in production is to produce an artefact of appropriate quality. This is also one of the main objectives in the development of computer based information systems (which we refer to hereafter simply as systems). During its short history, systems development has been affiliated with productivity problems such as slipping schedules and cost overruns (Brooks, 1975; Boehm, 1987), undisciplined development practices (Humphrey, 1989), and low systems quality with increased maintenance costs (Boehm, 1987). At the same time, many studies (Boehm, 1987; Charette, 1986; Turner, 1987; Flood and Carson, 1990) have noticed the increased complexity of systems, which poses new challenges for improving systems development. When trying to reach the objective above – to produce quality systems faster – neither improving *product* quality nor improving *process* quality should be ignored (Heineman et al., 1994).

Over the years, computer aided tools have been introduced to facilitate systems development tasks. The first tools were aimed at performing or automating separate and well-formalised tasks, e.g. compilers. Work on introducing single software engineering tools into integrated environments (SEEs) commenced in the mid 1970's (see history in Brown et al., 1992, for example). During the same decade the first tools to facilitate system analysis and design were introduced (see Bubenko et al., 1971) culminating in the birth of the first integrated modelling environment *PSL/PSA* (Teichroew and Hershey, 1977). During the 1980's the industrial emphasis on systems modelling tools has expanded, introducing the idea of the Computer Aided Systems Engineering (CASE) environment (Charette, 1986; Chikofsky and

Rubinstein, 1988; McClure, 1989). Similarly, the aim to find new architectural solutions for integrating software engineering tools yields the concept of Integrated Project Supporting Environment (IPSE) – a framework of common services, which can be populated with sets of tools to suit different software development needs (Dowson, 1987a; LeQuesne, 1990; Brown et al., 1992; Brown, 1993). During the 1990's research on tool architectures has introduced integrated CASE (ICASE) environments (Chen and Norman, 1992), process centred software engineering (PCSE) environments (Finkelstein et al., 1994; Garg and Jazayeri, 1996), and Software Factories (Fernström et al., 1992).

The history above shows the progress of building technically more advanced design environments, rather than addressing the problems of their suitability to organisations' development practices. In many organisations adaptation of a new design environment has not succeeded well, and the application of design environments has remained low. This is often due to the complexity of design environments and lack of training (Iivari, 1996). The gap between organisations' local methods and those supported in design environments has led to the rise of *customisable design environments*, in which local methods can be defined and incrementally modified. Despite the familiarity of methods in customisable design environments the complexity still exists raising the question "How can design environments be used effectively?"

In their study on design aid technology Henderson and Cooprider (1994) distinguished three aspects of support: *production, co-ordination* and *organisational* technology. Production technology (including representation, analysis and transformation of products) aims to impact the capacity of an individual to generate planning or design decisions and subsequent products. Co-ordination technology, which is further divided into control and co-operation functions, enables interactions of multiple actors and covers their rules, priorities and policies in the design process. Organisational technology, then, determines the context of organisational procedures and policies (help, learning material, organisational knowledge and rationales) to be embedded into the production and co-ordination process. The purpose of the organisational functionality is to partially answer the question above: guidance and human understanding as contributors to effectiveness. Henderson and Cooprider (1994) and Vessey and Shravanapudi (1995) report that most design environments succeeded well in supporting production activities of individual users, but their capabilities for supporting co-ordination and associated organisational functions are low. We see the comprehensive support of each function as essential when building new architectures for design environments.

This study focuses on customisable design environments that cover the support for both design artefacts and the design process. We take the perspective of organisational technology above and examine solutions for guiding design tasks. This study suggests a solution for future design environments, especially tackling the question of how to flexibly model the design process. Next we characterise the background for customisable design environments: *customisable systems modelling support for multiple users guided by a customisable process modelling approach.*

Modelling is one of the cornerstone activities in production technology, and models form the main artefacts produced by design environments. Traditional structured methods such as ISAC (Lundeberg et al., 1980) and Yourdon's Structured Analysis (Yourdon, 1989) as well as object-oriented methods such as OMT (Rumbaugh et al., 1991) and FUSION (Coleman et al., 1994) are heavily based on system modelling and on the use of system models. This is also seen in various method comparisons (e.g. Olle et al., 1982; Olle et al., 1983; Olle et al., 1986; Hong et al., 1993; Iivari, 1995). Because of their central role, wrongly selected modelling techniques may be an obstacle to producing high quality systems. Thus, it is beneficial for a design environment to be able to customise its modelling concepts and notations.

To clarify the various aspects of co-ordination Vessey and Shravanapudi (1995) classify design environments into three groups – *taskware*, *teamware*, and *groupware* environments – in which the latter extends the functionality of the former. Taskware environments are single user environments supporting the functions of production technology and storing design models into a single user repository. Teamware and groupware support multiple users and apply features of co-ordination technology. Teamware supports a group working toward a common goal, which is managed by sharing design products either synchronously or asynchronously. Groupware, then, is facilitated with explicit communication channels (like e-mail and brainstorming facilities), in addition to the indirect communication through design models. The use of current repository technology and client-server architectures will enable design environments to become teamware and the use of CSCW (Computer Supported Collaborative Work) facilities to become groupware, respectively. As working practices change when replacing taskware with teamware we need to deal with questions such as *how* to manage the development process?, *how* to sequence and synchronise tasks and assign them to developers?, and *how* to share design information between developers?

During the last decade interest in modelling and visualising system development processes (and also other working processes) has been high. We can notice this in several process modelling approaches (Curtis et al., 1992; McChesney, 1995; Heineman et al., 1994), and in the rise of the modelling approaches concerning workflow management (e.g. Chroust and Benczur, 1994; Joosten et al., 1994) and business processes (e.g. Scheer, 1994). Process models have various purposes, from human understanding to automation (Curtis et al., 1992). In this study we regard process models as part of the organisational technology discussed by Henderson and Cooprider (1994). The aims of such process models are to provide understanding of systems design as a process, and to guide and manage the use of both production and co-ordination functions, which are already provided by the design environment in concern.

We address two technologies: *Computer Aided Systems/Software Engineering (CASE)*, which provides modelling support in design environments, and *Process-Centred Systems/Software Engineering (PCSE)*, which covers process modelling support. Current PCSE approaches will be examined from the

viewpoint of integrating process support into design environments[1]. Research on both CASE and PCSE technologies spread in the late 1980's. Table 1.1 introduces some series of annual workshops and conferences. The next two Sections (1.1.1 and 1.1.2) discuss our findings on CASE and PCSE technologies, and Section 1.1.3 presents our approach with respect to them.

TABLE 1.1 Annual workshops and conferences focusing on CASE and PCSE

Acronym	The first publications in the series of workshops or conferences
CASE	International Workshop on Computer-Aided Software Engineering, Index Technology Corp., Boston, MA, 1987, (since 1992 published by IEEE Computer Society Press)
NGCT	Workshop on the Next Generation of CASE Tools (NGCT), (eds.) S. Brinkkemper and G. Wijers, Noordwijkerhout, the Netherlands, 1990.
ISPW	Software Process Workshop, (ed.) C. Potts, IEEE Computer Society Press, 1984 (since 1987 International Software Process Workshop)
ICSP	International Conference on the Software Process, (ed.) M. Dowson, IEEE Computer Society Press, October 1991.
EWSPT	Software Process Technology, (ed. J.C. Derniame), Lecture Notes in Computer Science #622, Springer-Verlag, 1992. (Proceedings of the Second European Workshop on Software Process Technology, Trondheim, Norway. The series of workshops started in Milan, 1991)

1.1.1 CASE Technology: Support for Production and Co-ordination

The aim of CASE technology is to automate or support parts of the systems development process, and thus increase systems quality and productivity. The requirements to increase both the quality (Wijers and van Dort, 1990; Aaen et al., 1992) and productivity (Norman and Nunamaker, 1989) have been laid out. However, the first attempts of CASE technology to empower organisations to deliver systems on time and to decrease development costs were not successful (Huff, 1992; Finlay and Mitchell, 1994). One explanation proposed is the low CASE usage (Iivari, 1996). Multiple theories (stemming from organisational, individual, political, and technical reasons, among others) have been brought forth to explain the low usage of CASE environments. These include lack of management support and poor training of staff, earlier experiences on CASE,

[1] In this study we use the term *design environment* as a synonym for a *CASE environment*. The reason is that design environment is a term without any architectural burden. Some authors (e.g. Merbeth, 1997) use CASE to denote an old-fashioned architecture based on a set of model translations. Design environment is a general term for any environment supporting design aid functions (production, co-ordination, and organisation) discussed in (Henderson and Cooprider, 1994). Traditionally CASE environment has been more restricted to modelling environments, but in Chapter 2 (Requirements for future CASE) we have extended it to include other functions of design aid technology also.

and the complexity and relative advantage of the CASE environment (Orlikowski, 1988; Iivari, 1996).

This study addresses CASE environments from the technological point of view, not forgetting related organisational issues. We base this study on two assumptions: 1) *method customisation* helps organisations' processes to adapt CASE technology, and 2) *teamware* environments facilitate the work of multiple agents. Among the questions raised by these assumptions are: how well can a CASE environment support organisations' local methods and practices?, how well can a CASE environment be customised to future method needs?, and how effectively can CASE environments support systems design by multiple agents? Focusing on the *method customisation* and *co-ordination* support of current CASE environments we can observe that – when present together – only partial solutions had been provided.

Method customisation support. Traditional CASE environments basically supported a single method, and their architecture made the accompanying method hard to modify. The method was, therefore, imposed on a user organisation through the introduction of a CASE environment. Empirical studies (Orlikowski, 1988; Aaen et al., 1992) show that organisational problems are fewer when a CASE environment 'matches' the method already in use. One prominent explanation brought up has been the difficulty of learning a new method and the use of a CASE environment simultaneously. Such a process often fails to succeed. Methods are rarely used as specified in textbooks (Chikofsky and Rubinstein, 1988; Wijers and van Dort, 1990), and local methods are widely used (Yourdon, 1992; Fitzgerald, 1995; Russo et al., 1994). In particular, changes in modelling concepts (i.e. design ontology) are difficult to carry out in an organisation (Jarke et al., 1998). To summarise, limited integration of methods and techniques, and inflexibility in choosing their representations have been observed as obstacles to effective CASE use (see also Lehman and Turski, 1987; Bubenko and Wangler, 1992).

One proposed solution is to provide support for method customisation by *metaCASE environments* (also called *CASE shells*), which contain mechanisms for defining and modifying an arbitrary method, and provide CASE tool support for the modelled method (Bubenko, 1988; Brinkkemper, 1990; Alderson, 1991). Until recently, metaCASE environments have been configured to provide their method support by importing a set of structured textual files and compiling them (see Appendix 2).

Besides capabilities to define methods, we need principles and tools for constructing methods to suit projects' needs precisely. For example, knowledge of the methods' use in earlier projects is suggested to improve selection and customisation of methods in future projects (Kumar and Welke, 1992). The discipline that studies the method construction and evolution process according to project factors is called *(situational) method engineering* (Kumar and Welke, 1992; Harmsen, 1997), and supporting tools are called *method engineering tools* (Harmsen and Brinkkemper, 1994b).

In general, metaCASE environments cannot compete with fixed CASE environments in the support of all detail of textbook methods[2]. For example, fancy layered aggregate symbols are hard to support. However, the benefits of metaCASE environments are seen in the following cases: (1) if there does not exist any CASE support for the specific method, (2) if an organisation is using several methods, (3) if an organisation is testing and selecting methods or building new ones, (4) if the future method requirements are fuzzy or evolving, and (5) when local variations of the textbook methods are used (Bubenko, 1988; Brinkkemper, 1990).

Co-ordination support. CASE environments mainly concentrate on a single user's process, i.e. creation, manipulation, analysis and transformation of models from the individual developer's viewpoint (Henderson and Cooprider, 1994; Vessey and Shravanapudi, 1995). This was also recognised in early metaCASE environments we compared (Marttiin et al., 1993). However, many studies (Curtis et al., 1988; Vessey and Shravanapudi, 1995) consider group activities like co-ordination and communication more critical for the project's productivity than individual ones. Although CASE environments and IPSEs were built primarily to support individual developers' tasks (LeQuesne, 1990), requirements have changed because of other technological innovations, e.g. multi-user repositories and collaborative applications (see Chen and Norman, 1992). Thus, new solutions to support group process in future CASE have been sought (Norman et al., 1991; Hahn et al., 1991; Forte and Norman, 1992; Vessey et al., 1992; Henderson and Cooprider, 1994; Jarke et al., 1994; Vessey and Shravanapudi, 1995; King, 1996; Grundy and Venable, 1996).

Co-ordination technology attempts to manage dependencies between tasks, which are carried out by multiple users and supported by a variety of tools. As Malone and Crowston (1994) say "good co-ordination is nearly invisible and problems and issues related to co-ordination become apparent when it is flawed". In this respect the aim of co-ordination technology should be keeping co-ordination invisible. A suggestion to aid co-ordination is the use of a shared information space and a system that contains information about different roles and procedures within a group or organisation (Grudin, 1991). Co-ordination benefits in CASE technology are related to sharing of design information – system models, model elements and related comments and arguments – between project members (Vessey and Shravanapudi, 1995).

1.1.2 Process Modelling in PCSE Environments

The software engineering community holds the view that the quality of products depends on the quality of the production process (see e.g. Humphrey, 1989; Armenise et al., 1993). Suggested benefits of process structuring include improved productivity through standardisation, and easier division of labour by breaking processes down into tasks (Fitzgerald, 1995). Besides, a well

---

[2] *Select OMT* is an example of a CASE environment built to support one fixed method – OMT (Rumbaugh et al., 1991)

defined and documented system development process can lay foundations for improving long term productivity (Humphrey, 1989; Curtis et al., 1992). This strong belief has led us to examine existing process modelling approaches built in PCSE environments, and study their suitability for supporting design processes carried out with the help of (meta)CASE environments.

We are looking for process modelling approaches that provide support for diversified production and co-ordination tasks in customisable design environments. We claim that essential features of such process modelling support are: 1) to provide *human understanding and guidance* and 2) to *customise process models* to satisfy the local process practices and process concepts used in organisations.

Human understanding and guidance. Purposes for process modelling are manifold and include: facilitating human understanding and learning, improving process and project management, providing automatic guidance, and process automation (Curtis et al., 1992). Regardless of the purpose, a process model is of low value if it is not understood by the developers that need to enact or use it. As noted by (Heineman et al., 1994) "even simple processes quickly become complicated because there can be many processes performed simultaneously by different people". Descriptive life cycle models have long been used for understanding and communication purposes. However, in PCSE environments this interest has received far less attention than prescriptive process support automation (McChesney, 1995). Many PCSE environments are following strictly defined process models, which often are described by using a formal process modelling language (PML) (see, e.g. Curtis et al., 1992; Armenise et al., 1993; McChesney, 1995).

To enact an automated process a small-grained and precise process model is necessary (Curtis et al., 1992). However, the feasibility of applying automation based approaches to design (concerning intellectual, uncertain and fuzzy tasks) can easily be criticised, as Lehman (1987) did ten years ago. When building process support into design environments, we need to solve the problem of providing simultaneously descriptive, understandable and less precise models for humans, and prescriptive and strictly defined models for tools.

Mostly process models are enacted through context specific menus or by an explicit process model. For example, in *Pro-Art* (Pohl, 1996), the future process is guided via context specific menu selections, and a graphical model describing the past process is created. Many PCSE environments contain a graphical process model as a user interface for humans to enact processes. Such environments include *STATEMATE* (Kellner and Hansen, 1989), *ProcessWeaver* (Fernström et al., 1992; Christie, 1995), *SPADE* (Bandinelli et al., 1993), *EPOS* (Jaccheri and Conradi, 1993), *Softman* (Mi and Scacchi, 1992), and *MENTOR* (SiSaid et al., 1996).

Process customisation. The idea in metaCASE environments is to support a range of methods for various organisational needs. Similarly, an organisation may want to define process support for its own specific needs. Other needs for an arbitrary process model are derived from the following two statements. First, systems development is a unique and uncertain process, and we cannot

predict all future tasks and define the precise order of them beforehand. This calls for evolution of process models by adding, changing or removing tasks and dependencies between tasks during a project (Conradi et al., 1993; Lonchamp, 1994; Kontio, 1995). Second, process models cannot capture all process perspectives; rather, they take a special process perspective to determine what 'things' a process model can represent (Curtis et al., 1992). Process models are human artefacts and their perspectives should match the needs of the project. Thus, there exist demands to also customise the conceptual structure behind process models (i.e. process metamodel), which further determines the suitable PMLs to represent process models. The discipline of managing customisation of both process models and their conceptual basis is called here *process engineering (PE)*. PE contains all activities to evolve or improve the software process (Lonchamp, 1993; Conradi et al., 1993). This set of activities is also called the meta-process, as distinct from the production process.

In recent years management of process model customisation has been one of the topics of interest in PCSE technology (see e.g. Madhavji, 1991; Fuggetta, 1993; Lonchamp, 1993; Jaccheri and Conradi, 1993). A set of PCSE environments to support process evolution will be presented in our survey (Chapter 4). Mechanisms to facilitate process model evolution, such as late/dynamic binding and model rebuilding mechanisms, are discussed in (Conradi et al., 1993). They say that on-line process model customisation will benefit from using programming languages that include late/dynamic binding, like Smalltalk. We have tested Smalltalk's binding mechanism and also found it useful for customising process perspectives (process metamodels), which can be managed well along with process model customisation.

Customising a PML to be enacted by tools and interpreted by process engines has been studied. The studies include the metalinguistic approach in Amber (Kaiser et al., 1993; Kaiser et al., 1996) and language extension mechanism by Balzer and Narayanaswamy (1993). These approaches, however, stress the customisability of the environments rather than focusing on defining process metamodels to specify process concepts interpreted by humans. PCSE environments to provide several perspectives for humans are mainly built on fixed process perspectives. Examples of such environments include *STATEMATE* (Kellner and Hansen, 1989) and *ALF* with the MASP language (Canals et al., 1994). Environments allowing the customisation of process perspectives are rare. EPOS, SPADE, and *CPCE* (Lonchamp, 1994) can adjust their process concepts by specifying them, but they still allow one process perspective for users. Closest to our interests is *OVAL* (Malone et al., 1995), which can customise concepts for co-ordination, and define a variety of co-ordination environments.

1.1.3 About This Study

As we noticed in Section 1.1.1, design environments can be customised to support organisations' local methods through modified techniques. In their shift to teamware, design environments need to apply co-ordination technology (Henderson and Cooprider, 1994). Organisational technology (i.e. building infrastructure for using design environments and support, learning aids and help in using design aid functions) becomes important, when we consider both the production and co-ordination functions. We take an approach in which most production and co-ordination functions are implemented in the design environment *MetaEdit+* (Kelly et al., 1996). In this study we suggest augmenting these functions with customisable process models and process modelling tools. Thus, process support is considered a part of organisational technology. The roles of MetaEdit+ and the implemented customisable process modelling environment (CPME) are discussed in Chapter 6 of this study.

Before going to the detailed research objectives and questions we will clarify the focus of our study in relation to the architectures of design environments, process integration, and granularity level of process models.

Architectures of design environments. One main question is how to build a multi-user architecture to be applied in a variety of situations. Curtis et al. (1992) presented artefact (i.e. products, deliverables), task (i.e. process) and agent (i.e. actors and tools) as the most prominent elements for process modelling. These three elements also express basic elements in the definition of method support: techniques, design process, and agent behaviour to be stored in a repository of a design environment. By customising such a repository schema we can customise a design environment to fit an organisational context more intensively than environments focused on defining pure techniques. All three elements are integrated in our suggestion for method support in future metaCASE (see Chapters 2 and 3). We stress the study of the relationship between the product and the process.

Process integration. In the suggested approach we are not developing a process centred design environment, in which all *task*s are interpreted and executed by a process engine. Our view of design environments and their architectures (see Figure 4.1 in Chapter 4) differs from PCSE environments based on process engines. Process models provide additional guidance for existing production and co-ordination functions of a design environment. In this respect, similar approaches include the process-driven CASE environment *Softman* by (Mi and Scacchi, 1992), and *HyperCASE* (Cybulski and Reed, 1992) providing 'additional' hypertext functionality for a CASE environment. However, these environments include sets of loosely coupled tools and the role of process integration is stronger than in our case, where design information is already integrated together in a common repository. Our interest is more on customisable solutions for process models and process metamodels than the integrity of design information.

The functionality of process models. We consider process models as a means to human understanding and guidance. When visualising process models graphically, they can be used to aid common understanding of the system

design process. A process model decomposes and sequences the modelling *tasks* and manages their dependencies. A process model is, firstly, used to guide developers through tasks they are obliged to perform and aid the collaborative design process by co-ordinating the work of multiple users. Guidance means messages, feedback and information about the current situation. Secondly, a process model can co-ordinate the sharing and manipulation of design products. These products are accessed by developers through *tasks* of process models. A process model forms a user interface for users to join the design process, i.e. to access products through enacting *tasks*.

The granularity in process models. To clarify the granularity of the study, we present systems modelling and process modelling at three levels (Table 1.2). On the project level the focus is on *life cycle stages* to produce *products* (e.g. production of system requirements). Correspondingly, at the technique level *modelling tasks* are expected to produce *models* (e.g. consistency checking of a model). On the model level we have tasks to manipulate *modelling elements (model manipulations)* (e.g. creation of an entity).

TABLE 1.2 Granularity Levels in System Modelling and Process Modelling (Verhoef, 1993)

Granularity levels	*System modelling*	*Process modelling*
project (method)	products	life cycle stages
technique	models	modelling tasks
model	modelling elements	model manipulations

Our focus is on the technique level (greyed in Table 1.2). The deliverables we are interested in are models. The process model serves as an interface for users to enact the modelling tasks. Process models co-ordinate users' access to produce or to manipulate a model relevant for the activity.

To have a better understanding of the process as a whole modelling tasks should be integrated into descriptive life cycle models. Obviously synergy can be gained when the modelling tasks of a process model in a CASE environment and a process model in a quality or maturity approach are united.

Model manipulations in CASE environments are often hard coded and not guided by a process model (Vessey et al., 1992). In MetaEdit+ the model manipulation tasks of each technique are managed uniformly. A process support *forcing* developers to follow a fine-grained process has not been seen as feasible in metaCASE because of the differences between modelling styles (Verhoef and ter Hofstedte, 1995). Undoubtedly, CASE editors can be built to allow developers to choose from a set of appropriate tasks based on a process model, such as the context based menus provided by Pro-Art. Because our approach is an extension to MetaEdit+ (see *process integration* above) this study does not cover model manipulation issues.

1.2 Research Objectives and Questions

Two main objectives of this thesis are: (1) *to widen the understanding of customisable design environments*, and (2) *to study and build the process support that facilitates the use of a customisable design environment*. A main assumption in this study (discussed in Section 1) is that systems development is dependent on the situation, and thus it requires customisation of methods and process models for various needs and situations. This research concentrates on customisable design environments called metaCASE environments.

MetaCASE environments have to date focused on the tasks of an individual developer (taskware). In these environments method support has focused on defining production support: modelling techniques, integration between techniques, and related analysis and transformation support. Currently CASE environments are moving towards multi-user design environments (teamware) with increased co-ordination functions. Because group work is a complex activity, the architectures of design environments are at the same time becoming more complex. According to Henderson and Cooprider (1994), production and co-ordination functions will benefit from organisational support such as additional guidance on how to use the design environment in an organisation. For this reason, we integrate a process model into production and co-ordination functions. The general research problem related to this situation can be formulated as follows:

> How can we develop process modelling support to facilitate production and co-ordination functions of customisable design environments?

The first of our objectives *widening the understanding of design environments* motivates the writing of an introductory chapter, which will present basic concepts and the research area. We focus on two levels: systems development on the lower level, and method engineering (ME) and process engineering (PE) on the higher level (Chapter 1, Sections 2 and 3). The research question related to the first objective is:

> **Question 1** What has been the focus of method support in metaCASE environments? (Chapter 1, Sections 2 and 3)

To answer the first research question we gathered experiences of method support in metaCASE environments (Marttiin et al., 1993, see Section 2.1.2, and Appendices 1 and 2). Although our laboratory study, in which we modelled the SMARTIE method in three metaCASE environments, did not focus on process aspects the need for an all-inclusive and uniform conceptual basis for future metaCASE environments was evident. In addition, the study shows our discontent with the present method engineering tools.

To relate our experiences in a larger context we needed to have an understanding of the phenomena of ME/PE. An overview of this research is discussed in Section 2.2. The question of what kind of ME support is

appropriate is not well understood. Accordingly, in Section 3 we apply the model of design aid technology (Henderson and Cooprider, 1994) into ME. Although we can not prove the feasibility of the framework it shows limited support for ME aspects in current design environments. We can also notice that method specifications defined in CASE environments tackle mostly production functions.

Our second objective is divided into two sub-objectives, which are (a) *to design a conceptual basis for a customisable multi-user design environment*, and (b) *to develop customisable process modelling tools to guide the use of a design environment*. Results tackling these sub-objectives will be discussed in 5 research papers forming the second part of the thesis (Chapters 2 to 6). The research questions focusing on the second objective are:

> **Question 2** How to design a conceptual architecture for a metaCASE environment that enables the customisation of method support for multiple users? (Chapter 2)
> **Question 3** What are meaningful process modelling principles that support systems development with design environments? (Chapter 3)
> **Question 4** How customisable are process models and their conceptual basis in current PCSE environments? (Chapter 4)
> **Question 5** How to build customisable process support that supports the production and co-ordination functions of a metaCASE environment? (Chapters 5 & 6)

Each research question is derived from tool evaluations and requirements found from the current literature on systems development methods, process modelling, (meta-)CASE and PCSE environments. The resulting ideas of the thesis are demonstrated by developing new kinds of tools according to the objectives of the study. We designed and built process modelling tools to support and guide the use of metaCASE. We consider process modelling as a means for potential productivity improvements; those improvements remain conjectures in this study.

The second question focuses on the design of a metaCASE environment. Chapter 2 provides a wide range of requirements for method aspects to be modelled and defined in a metaCASE environment. We build a conceptual basis for a metaCASE environment, which tackles aspects of meta-data (modelling techniques), process and agents.

In Chapter 3 we concentrate on designing process support for metaCASE, and study processes in relation to modelling techniques and participating agents. We focus on the question of meaningful process modelling principles applied in metaCASE. As a result of this study, we consider a process model primarily as a means to facilitate human understanding and guidance (as discussed in Section 1.1.3).

Process modelling approaches have been developed for several engineering domains. When designing process support for the domain of multi-user metaCASE we can not ignore multi-user PCSE environments. In Chapter 4 we make a survey of current PCSE environments and examine the suitability of

their process approaches for design environments used in evolving situations. The characteristics of process (uncertainty and situation dependency) require process customisation and may demand changes in process metamodels.

The last question tackles the problem of implementing the process support environment we call Customisable Process Modelling Environment (CPME). It provides process modelling support for MetaEdit+. This prototype is based on the current architectural solutions of MetaEdit+. Chapter 5 presents the process metamodelling language and tools for defining the conceptual structures behind PMLs (process metamodels). It shows how the example process metamodel, VPL (Shephard et al., 1992), can be defined. Although we selected task based PML to be an example it does not prevent us to define and use other PMLs e.g. a decision based PML (Rolland et al., 1995). Chapter 6 presents process models in relation to MetaEdit+ production (meta-data) and co-ordination (agents), and introduces the Process Editor tool for creating and enacting process models. We show the implemented parts of the future metaCASE environment presented in Chapter 2. Chapter 6 presents a process of creating a PML and a process model to guide and co-ordinate design tasks. We use VPL to model the ISPW-6 example process (Kellner et al., 1991).

1.3 Research Methodology

Wynekoop and Conger (1991) made a survey on methodological approaches used in CASE research over the years 1987–9. They observed that "the purpose of the research determines, and is determined by, the methodology used", and classify the CASE research purpose into the following classes: understanding/ description, engineering, re-engineering and evaluation. They examined the basic research methods used in CASE. Literature surveys and normative writings are mostly used for increasing understanding and describing ideal CASE characteristics. Applied research such as system building is used in re-engineering/engineering, and case studies and surveys in evaluation.

We have selected system building as our research approach. System building as an IS research methodology has only been discussed quite recently, although it has long been used in computer science (Denning et al., 1989). It is ignored in older studies (e.g. Benbasat, 1984), but mentioned in Galliers (1991). Järvinen and Järvinen (1993) discuss constructive study as a creation of new knowledge, and system building as a means to new ways of working. A detailed discussion of system building as a research approach is found in (Nunamaker et al., 1991), and it is this which is followed in this study.

In the discussion of a research methodology (Nunamaker et al., 1991; Nunamaker, 1992) a multi-methodological approach for conducting constructive scientific work is outlined. It consists of theory building, experimentation, observation, and system building. "The relevance of a theory refers to its potential to suggest insights on issues that have not yet been explored. ...it supports and it is supported by system building, and is both a

precursor for and outcome of experimentation and observation" (Nunamaker, 1992, p.3). Experimentation as a testing of the hypotheses of the theory contains laboratory tests, field experimentation, and simulation. Observations include case studies and surveys, which are used when little is known of the research domain.

According to Nunamaker et al. (1991), tool development is an incremental and iterative process where knowledge accumulates during development. It contains the following steps: framework construction, system architecture development, analysis and design of the system, prototype building and evaluation. From each step we can return back, for example, to ask a more detailed research question or to change an architectural principle. To justify the use of tool development as a research approach the research phenomenon should conform to the following five criteria:

1. the phenomenon is important,
2. the results make a significant contribution to the domain,
3. the tool is testable against all statements and requirements,
4. the tool provides better solutions than existing tools, and
5. the experience and design expertise can be generalised for future use.

The following section describes the research process and how the five criteria are conformed in this study.

1.4 Research Process Following a Constructive Research Methodology

1.4.1 Research Process and Publications in a Chronological Order

This section describes the research process of this study. It can be divided into periods each linked to separate research projects at the University of Jyväskylä: in SYTI, MetaPHOR, and CAMSO projects.

Directing research to study meaningful questions requires some background studies. One background study for this research has been the participation in the SYTI project (funded by TEKES) during the years 1989–91. We developed generic principles for a metaCASE tool, analysed, designed, and built a prototype called *MetaEdit*[3] (Smolander et al., 1991; Marttiin, 1991; Rossi, 1993). We did a self evaluation of the MetaEdit functionality and compared it with other metaCASE products and research prototypes (Marttiin, 1991), and got responses from MetaEdit users in industry. In other words, we went once through the research life cycle discussed in Section 1.3. The main advantages of

[3] Later on this prototype was developed to a product called MetaEdit™ Personal 1.0 by MetaCase Consulting.

MetaEdit were its simple conceptual basis (OPRR, defined in Smolander, 1992) and easy-to-use tools to customise modelling techniques. However, it was limited to supporting one technique at a time.

An answer to the first research question *What has been the focus of method support in metaCASE environments?* is found in the tool comparison, which we carried out as a laboratory experiment with three metaCASE environments (see Marttiin et al., 1993). We built a comparison framework for metaCASE environments, and compared them in terms of their functionality and usability. This study concentrated on modelling techniques and defined the SMARTIE concepts and representations. These tools were based on single-user repositories, and they lack process guidance. These three metaCASE environments are presented in Section 2.1.2 with Appendices 1 and 2.

The first and second papers of this study have been written in the 1991-3 MetaPHOR project (Lyytinen et al., 1994) funded by the Academy of Finland. We started the project with a normative study 'Modelling Requirements for Future CASE: Modelling Issues and Architectural Considerations' (Marttiin et al., 1992; Marttiin et al., 1995; Chapter 2) addressing requirements for a metaCASE environment. As a main result a metaCASE architecture integrating modelling techniques, the development process and participating agents was formed in this study. We also tested the conceptual model by defining the IBA/SIM method (Tolvanen et al., 1993) and several object-oriented methods (Tolvanen et al., 1992). The second paper 'Towards Flexible Process Support with a MetaCASE Environment' (Marttiin, 1994; Chapter 3) examined principles for process support in metaCASE. In this paper we suggested a process oriented approach for co-ordination in metaCASE. These two papers discuss the research questions *How to design a conceptual architecture for a metaCASE environment that enables the customisation of method support for multiple users?* and *What are meaningful process modelling principles that support systems development with design environments?* During the years 1994-6 I participated in the CAMSO project (Lyytinen et al., 1997), which was supported by the Ministry of Education and the Academy of Finland. The main components of the *MetaEdit+* metaCASE environment (Kelly et al., 1996), into which our process modelling environment will be integrated, were developed during the years 1994-5. During the summer of 1995 the first prototype for defining process metamodels was implemented as a Master's thesis (Koskinen, 1996). This prototype is introduced in the fourth paper of this study: 'Process Support in MetaCASE: Implementing the Conceptual Basis for Enactable Process Models in MetaEdit+' (Koskinen and Marttiin, 1997; Chapter 5), which partly answers Question 5 *How to build customisable process support that supports the production and co-ordination functions of a metaCASE environment?*

During the autumn of 1995 I visited the University of Twente in the Netherlands. We built a framework for metaCASE and computer aided method engineering applying the functional model of Henderson and Cooprider (1994). This framework is applied in the evaluation of two metaCASE environments: MetaEdit+ and *Maestro II* with *Decamerone* (Marttiin et al., 1996). We use the framework to show the research done and to find unanswered questions in the

fields of metaCASE and method engineering. This two level framework will be introduced in Section 3 of this chapter.

New environments have been designed and/or documented during the period we have built MetaEdit+ (see e.g. SiSaid et al., 1996; Pohl, 1996; Grundy and Venable, 1996; Taivalsaari and Vaaraniemi, 1997). The purpose of the third paper 'Can Process-Centred Environments Provide The Customised Process Support Needed in MetaCASE? A Literature Review' (Marttiin, 1997; Chapter 4) is to show the variety of process theories and PCSE environments based on these theories. Furthermore, we examined how these solutions support the process modelling requirements derived from metaCASE and, thus, how these can be suited to model design processes in a metaCASE environment. This will provide an answer to Question 4 *How customisable are process models and their conceptual basis in current PCSE environments?*

During 1997 we continued the development of the Process Metamodelling tools and developed a first version of the Process Editor to be used for both process modelling and process enactment. The Process Editor uses any PML defined with the Process Metamodelling tools. We call the set of integrated process tools Customisable Process Modelling Environment (CPME). To enable CPME to facilitate co-ordination, we made preliminary support for agent models and integrated products, processes and agents as presented in Chapter 2. The tasks of PML definition, project definition, process modelling, process enactment and process customisation are discussed in the paper 'How to Support CASE Activities through Customisable Process Models: Experiments on CPME/MetaEdit+ Using the VPL Formalism and ISPW-6 Example' (Marttiin, 1998; Chapter 6).

1.4.2 This Study as a Constructive Research Effort

In this section we describe how different chapters are related to the constructive research framework and how this study fulfils the criteria of the use of constructive research as a scientific methodology.

TABLE 1.3 This Study and the Framework by Nunamaker et al. (1991)

Chapters	Name	Nunamaker et al. (1991) framework
Chapter 1: Section 2	Research Area and Basic Terminology	Theory building, Experimentation: summarising a comparison of three metaCASE environments
Chapter 1: Section 3	Towards a Unified View on the CASE and CAME Technologies	Theory building
Chapter 2	Modelling Requirements for Future CASE: Modelling Issues and Architectural Considerations	Theory building, System building: requirement analysis, design
Chapter 3	Towards Flexible Process Support with a MetaCASE Environment	System building: requirement analysis, design
Chapter 4	Can Process-Centred Environments Provide The Customised Process Support Needed in MetaCASE? A Literature Review	Observation: a survey
Chapter 5	Process Support in MetaCASE: Implementing the Conceptual Basis for Enactable Process Models in MetaEdit+	System building: a prototype
Chapter 6	How to Support CASE Activities through Customisable Process Models: Experiments on CPME/MetaEdit+ Using the VPL Formalism and ISPW-6 Example	System building: a prototype. Experimentation: an example of PML definition, ISPW-6 example

Firstly, we present a cycle of constructive research including theory building, system building, experimentation, and observation. The introduction of this study reviews the literature of the phenomena and the reason why we are developing new kinds of tools. In addition, it presents our findings on the first metaCASE and PCSE environments. We build our theory based on contingencies and the requirement of customisability for methods and supporting technology (Sections 2 & 3). Chapter 2 provides a normative study to discuss requirements for future design environments (theory building). It also presents an example of how to design a customisable design environment fulfilling the requirements (system building). In Chapter 3 we take a process oriented view of the requirements and design process support for a future design environment (system building). Chapter 4 observes process modelling approaches of the 'state of the art' PCSE environments. It characterises the environments according to their abilities to define and customise process metamodels and process models (observation). Chapters 5 and 6 present the

CPME prototype we have built (system building). CPME consists of process metamodelling tools and a conceptual basis for defining process metamodels (called GOPRR-p), which are described in Chapter 5. Chapter 6 shows how the relationships between meta-data, process and agent models are implemented in CPME and MetaEdit+. Furthermore, it describes how to model process definitions, instantiate them for a project's use, and customise process models. Because Chapter 6 discusses the process of defining an arbitrary PML we can consider it as experimentation on CPME capabilities (process metamodelling language and process metamodelling tools).

Secondly, we discuss how the five criteria within the research methodology (Nunamaker et al., 1991) are fulfilled in the thesis.

Importance of phenomenon. The importance of customisable design environments has been noted in the discussion of the popularity of using in-house methods (see Section 1.1.1). Also, the importance of process modelling and the use of process models for human understanding and guidance is discussed (Section 1.1.2).

Significance of results. The contribution of this study is twofold: widening the understanding of the field and designing a customisable process modelling environment according to requirements for systems development and metaCASE technology. Most of the work in this thesis has been published in international journals and refereed conferences. Furthermore, MetaEdit+ – the basis we are building CPME concepts and tools – has been used and evaluated in industry. This will give us a possibility to test the usability of the results presented in this study.

Testability. The process modelling environment can be tested for customisability of both process models and process metamodels. However, the practical significance of the implemented solution – a process model as a guidance tool – can not be guaranteed without repeated observations and experiments.

Better solutions than existing tools. The support for customisation of process models and process metamodels has not yet reached sufficient maturity. Customisation of process metamodels is not available in any environment discussed in this study. A design of such a customisable process modelling environment and its integration into a metaCASE environment satisfies this requirement.

Generalisation of expertise. The metamodelling approach is already used to design other meta-tools such as DSS generators and compiler-compilers (see, e.g. Chen, 1988; Karrer and Scacchi, 1993). We have applied the metamodelling approach of MetaEdit+ straightforwardly in our process modelling environment by specialising existing MetaEdit+ primitives and tools. The process modelling approach we present in this study can be technically integrated into any design aid levels. For example, we can use the approach to help CAME activities in addition to the CASE functions discussed in this study. Implementation also supports the attachment of activities into other Smalltalk-based tools and deliverables than those of MetaEdit+. In this case the role of the process model changes towards process integration through loosely coupled tools.

2 RESEARCH AREA AND BASIC TERMINOLOGY

In this study we analyse, examine and develop tool support for system designers. The ultimate purpose of such tools is to facilitate system designers' work and improve the quality of produced systems and the design process. Therefore, in order to be able to develop useful tools we need to understand systems design as an essential part of (information) systems development, and the role of methods in this process.

Figure 1.1 shows the relationship between systems development, method engineering, and process engineering. *Systems development* is a disciplined approach to produce systems. It deals with concepts, notations, processes, goals, and agents determined by the method used. Respectively, *method engineering* is a disciplined approach to produce methods used in systems development. Parallel with method engineering there exists a process movement aiming at process quality and improvement. Improvements are facilitated by process modelling, monitoring and evolution, and they require *process engineering* approaches in order to be disciplined. Both method and process engineering may determine their concepts, notations, processes, goals, and agents, the focus being respectively on producing either methods or process models[4]. Although Figure 1.1 shows method and process engineering in separate boxes they are overlapping and complementary. In this study, we operate on the overlapping area of method and process engineering trying to complement supported methods with process aspects (process models) used to facilitate systems development.

---

[4] We use the term process model to denote both process definitions and instances of these (process model instances).

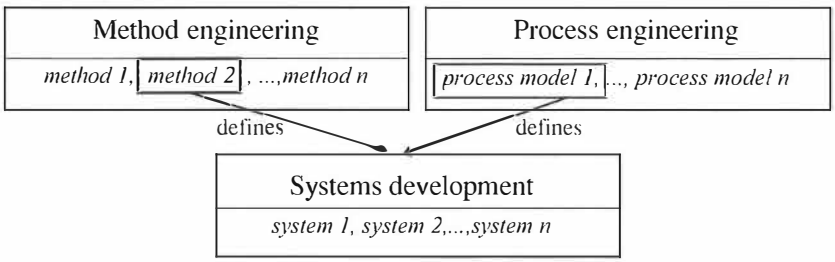

FIGURE 1.1 Systems development, method engineering and process engineering

The purpose of this chapter is twofold. Firstly, we introduce the basic concepts and definitions associated with systems development (Section 2.1), and method engineering and process engineering (Section 2.2). Secondly, we present the background for our research subject and discuss the assumptions and chosen theories in this study. Section 2.1 expresses the need for both method and process engineering, and Section 2.2 reviews current method/process engineering approaches.

2.1 Systems Development

Tools are always based on some assumptions of the phenomena and environment in which they will be used. Most of the tools we examined are built to be tackled on two research branches, information systems development (ISD) and software engineering (SE), which traditionally have differed in their assumptions about the development process. To clarify these differences we first discuss differences between ISD and SE.

Firstly, ISD focuses more on early tasks of life cycle such as requirements engineering, system analysis and design, whereas the interest of SE is to study development, operation, maintenance and retirement of software (cf. IEEE, 1983). SE mostly takes system requirements as given, while clearing up the system requirements has been one of the most problematic tasks in ISD. Secondly, SE considers software production a repeatable and measurable process, while ISD takes a one-time view of the process by also studying changes in the environment (e.g. changes in working conditions and organisational arrangements) where the software will be used (see Kumar and Welke, 1992). Although it is no longer appropriate to make such a separation between these two communities we would like to make it clear that we base this study on assumptions long attached to ISD: uncertainty and uniqueness. Thus, we rely on the importance of early life cycle tasks and the role of understanding and guidance during these phases. Design methods and design processes are situation dependent, justifying method and process engineering, and stressing the importance of tool customisability in method and process support.

Starting with basic concepts we characterise a system with an often used definition:

> **Definition 1** An (information) system consists of people, processes, data, models, technology, and language forming a structure which serves an organisational purpose or function (Davis and Olson, 1985; Hirschheim et al., 1995).

More clearly, the system consists of human systems (informal ones dealing with interactions between groups and individuals and formal ones built on rules and regulations), and computer/software systems (formal ones comprising automated activities and informal ones, e.g. passing messages and managing unstructured information). These system views always exist in relation to a surrounding environment (Land and Kennedy-McGregor, 1987). This composition makes room for a diversity of systems, thus bringing information systems farther away from systems that can be developed in an assembly line. Furthermore, the substance of each system is situation dependent: systems work differently depending on their environments and relations to other systems.

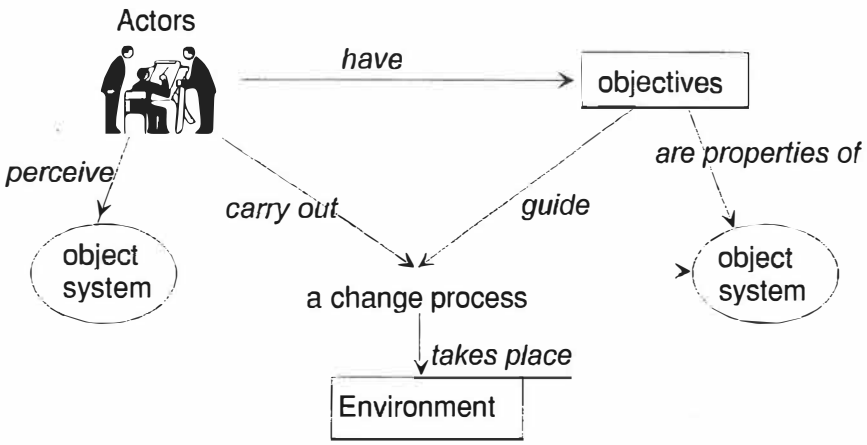

FIGURE 1.2 Systems development process (Lyytinen, 1987a; Hirschheim et al., 1995)

We consider systems development as a process transforming the original state of a system into a new one (Figure 1.2). Systems development as a change process has been discussed in (Welke, 1983; Lyytinen, 1987a; Hirschheim et al., 1995) and defined as follows:

> **Definition 2** Systems development is a change process, taken with respect to object systems, by a development group, to achieve and/or maintain one or more objectives (Welke, 1983).

We will notice the situation dependency of the systems development process when looking at the details of the definition. The *change process* can be described as an event where a development group acts on, formulates and alters current object systems guided by multiple objectives. An *object system* consists of phenomena, which the development group is observing. Object systems are perceptions of the target of change, which may vary among the members of the development group. Development takes place in an *environment* which (cf. Hirschheim et al., 1995) is formed by labour, economy, technology, and applications. The impetus for the process may result from beliefs, e.g. a produced system will improve working conditions or afford organisational success, or environmental changes, e.g. the development process may be justified by new technology. Finally, this *multi-objective* change process is marked by intentionality, inter-subjectivity and uncertainty (see Lyytinen, 1987a; Conradi et al., 1993; Kirsch, 1996).

Systems development methods define ways to conduct the change process. In practice, systems development has been carried out in a way which in most cases can hardly be called a methodical approach (Olerup, 1991; Fitzgerald, 1996). The selected approach is contingent upon the development situation. However, the increased use of development tools forces developers to follow systematic approaches: to measure processes, to collect design history and rationale, and to use specific modelling concepts and notations. We claim that all benefits of tools can not be achieved if the methodical approach that the tool provides is not followed in its entirety. At the same time, the benefits of tools are hardly achieved without a method suitable for the development situation. New tool adoptions may change old methods and development practices, but at the same time tools should be customisable to suit old practices. Therefore, the question of how a method suits a specific development situation and how a tool can support the specific method should not be asked separately.

Figure 1.3 shows prominent elements including *actors*, *design tools*, *methods*, *process models*, and *environment (development situation)*. Here, an *actor* is a general term for any human participating in systems development, and in the plural it is used to mean a project or a team. Another term we use in this study is *agent*, used for both humans (human agent) and machines (technical agent).

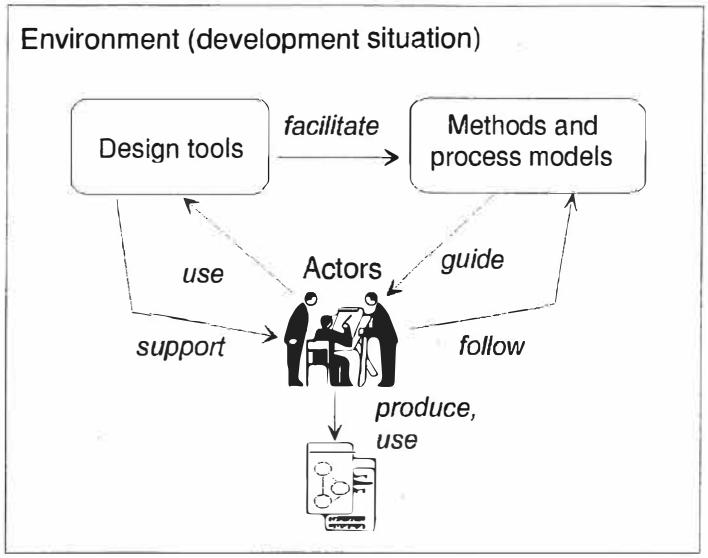

FIGURE 1.3 Design tools, methods and process models facilitating actors in design process

Design tools can support actors participating in the design process. They may automate some well-defined routine tasks, but their main purpose is to give support in the use of disciplinary approaches: design methods or a process model. In some tasks, however, the feasibility of tool and methodical support can be questioned. We can also talk about design (information) systems, and make a distinction (cf. Land and Kennedy-McGregor, 1987) between human and computer design (information) systems by dealing with the questions *What tasks require design tools?* and *What tasks can be better managed without design tools?* and between formal and informal systems by asking *What tasks require methods or process models to be followed?* and *What tasks can be better carried out amethodically?*

Systems development takes place in a situation which is a composition of different factors (also called contingencies): actors, methods, processes, tools, environments and relationships between these. These factors should determine the selection of reasonable design information system, thus determining what methods, process models, and supporting tools are to be used.

Thus, our interest is in modelling techniques and process models. Effects of any specific factor are not emphasised. Our view is that methods and process models as well as their tool support need to be customised for a specific situation. In the following subsections we look closely at these elements, starting from the methods and process models.

2.1.1 Systems Development Methods

System models can be used as a means to understand, communicate, learn, analyse, design, and predict (Olle et al., 1991; Hirschheim et al., 1995). Under these circumstances, models help developers to improve various quality

attributes, like functionality, reliability, usability, efficiency, maintainability and portability (IEEE, 1992). Modelling focuses especially on the functionality and reliability attributes, systems consistency, correctness, robustness, and validity.

As discussed earlier a system design *method*[5] defines ways to produce a system in a disciplined manner. This discipline is present in the definition by (Hirschheim et al., 1995):

> **Definition 3** A method is an organised collection of concepts, *techniques*, beliefs, values, and normative principles.

The number of suggested methods to govern the development process is vast and can be counted in thousands (Bubenko, 1986; Wijers, 1991, Avison and Fitzgerald, 1988; Jayaratna, 1994). Example methods include ISAC (Lundeberg et al., 1980), Structured Analysis (Yourdon, 1989), and OMT (Rumbaugh et al., 1991).

Some purposes of a method are to help communicate effectively, increase the understanding of the phenomena, analyse the current system, and specify the future system (Wasserman et al., 1983; Lyytinen et al., 1989). To guide how the change process can be carried out, methods may define (cf. Smolander et al., 1991) in what roles actors participate in development, in what order development tasks are accomplished, what development strategies are selected, and what organisational policies, principles, standards, and quality criteria are used.

Techniques form the core of a method determining how to advance in the change process. Technique is a way to accomplish a desired state of affairs by a series of steps (Welke, 1983). We can use modelling techniques to support systems analysis and design, questionnaires and interviewing techniques to gather information, brainstorming techniques in co-operation, and reviewing techniques to validate information, as examples. This study concentrates on modelling techniques such as *Data Flow Diagramming* (Gane and Sarson, 1979), *ER modelling* (Chen, 1976), and various object-oriented modelling techniques (e.g. Coad and Yourdon, 1991; Rumbaugh et al., 1991; Rational Software Corporation, 1997).

When discussing methods as 'formalisable artefacts' we focus on the aspects what can be defined and supported by tools. Kronlöf et al. (1993) distinguish the following aspects to be concerned: the underlying model (we prefer the term conceptual structure), description language(s), development steps, and guidance for applying a method.

During the modelling process a perceived object system is defined using a conceptual structure. A conceptual structure includes "a set of concepts and relationships forming a vocabulary to define system models" (Neches et al., 1991). It is also called an ontology, but, because some studies use ontology to

---

[5] We use the term method in the context of design method and methodology in the context of research methodology. According to Iivari (1991b) several methodological schools can be distinguished. Methodology can also mean a study of methods (see e.g. Checkland, 1981; Brinkkemper, 1990), or a science of methods (Baskerville et al., 1992).

denote a set of the very generic concepts and relationships for systems development (Wand, 1996) we use the term conceptual structure for clarity. Modelling techniques are based on a subset of the conceptual structure, and thus modelling may stress data, process, behaviour, organisation, problem etc. aspects of a system (Olle et al., 1982; van Slooten, 1995).

Description languages make the conceptualisation of the object system shareable among the members of the development group. They serve as a means of analysis, understanding and prediction of the structure and behaviour of a specific system aspect. As does any language, description languages contain syntax and semantics. Syntax defines the valid constructs of the language, and semantics associates a meaning for each construct. For example, mathematical semantics support exactness and analytical treatment, whereas operational semantics permit simulation and code generation (Kronlöf et al., 1993). Syntax can take various presentation styles including graphical, matrix, tabular, textual, symbolic or mixed syntaxes. Examples of graphical syntax include *data flow diagrams* (Gane and Sarson, 1979), tables are used in many lists, and text for example in *root definitions* (Checkland, 1981). Techniques can vary according to their degree of *formalisation*. In formal techniques the syntax and semantics are rigorously defined, semi-formal techniques contain syntactic rules, and in free form techniques there exists no complete set of rules to constrain the models created by techniques (Brinkkemper, 1990). In general, the degree of formalisation increases during the development life cycle. However, one should note that formal techniques such as *Z* (Spivey, 1988) are introduced to be used at as early a stage as requirement engineering.

In a conceptually consistent method each description language should use a common conceptual structure in order that techniques can reuse the already modelled system components. However, such a consistency is not found in all methods. Duplicate, overlapping and semantically fuzzy concepts make integration between techniques more difficult. Later on, in Chapter 2, we will discuss this issue in design architectures, and distinguish between horizontal and vertical integration. Horizontal integration exists between techniques on the same abstraction level such as integrating data and behaviour oriented design techniques. Vertical integration specifies a set of linguistic transformations from early stage models into detailed ones (Lehman and Turski, 1987), e.g. transforming an ER model into a database schema. Integration issues within a method and between methods are discussed in (Bubenko and Wangler, 1992; Kronlöf, 1993).

Methods include a *process* aspect describing tasks for actors to carry out. Many method books include process models describing sequences and concurrency of tasks related to the production of system models and documents (like in Yourdon, 1989). The granularity of the process models can be classified into life cycle models, modelling tasks and model manipulations (as seen in Table 1.2). The purpose of life cycle models is descriptive, while ordering of modelling tasks (model creations, analysis, and transformations) may be intended to be interpreted in a more prescriptive 'cookbook' style. Model manipulations, then, include the most elementary steps in model creation and are close to operational semantics giving the legal operations and supporting

automation. Sometimes lists of model manipulation tasks are used to guide effective model creation (e.g. various 'How to do' lists in Coad and Yourdon, 1991). Requirements for the process (how many times to repeat modelling activities and model manipulations) can be derived from quality standards to create complete, consistent, and accurate models (see Brinkkemper, 1990 pp. 42-46).

Method books normally provide a general 'cookbook' style of guidance. The most commonly used way to customise a method is providing additional guidance, like a casebook of examples (Kronlöf et al., 1993). A method can be applied to suit different domains better when organisation or domain specific guidance is introduced. In addition, the captured method knowledge and design rationale of earlier projects can be used when applying a method in a new project.

2.1.2 CASE and MetaCASE Environments

Computer Aided Systems Engineering (CASE) has been promoted as one of the solutions to assist in the development of quality products. Because various definitions and views of CASE abound (see Henderson and Cooprider, 1994), we do not try to suggest a clear and exact definition for CASE. We simply consider *CASE* a design aid technology. Following this definition a *CASE tool* is a design aid tool focusing on one prominent task and a *CASE environment*[6] is a collection of CASE tools covering the life cycle more comprehensively. Traditionally, the design aid provided by CASE technology has focused on systems modelling aspects, i.e. creation, analysis and transformation of system models (Henderson and Cooprider, 1994). CASE environments are standardisation and normalisation tools for methods and techniques, the main benefits of which are documentation and verification support, the integration of various models and representations, and specification and code generation (Wijers, 1991, pp. 25-28)

A comprehensive discussion on definitions and required tools and functions is found in early studies on CASE (e.g. Charette, 1986; Chikofsky and Rubinstein, 1988; McClure, 1989; Terry and Logee, 1990; Fuggetta, 1993), CASE frameworks (e.g. Misra, 1990; Crozier et al., 1989), and discussions on desired CASE functionality (Norman et al., 1991; Forte and Norman, 1992). The differences between CASE environments and Software Engineering Environments (SEE) and Integrated Project Supported Environments (IPSE) are discussed in (Brown et al., 1992, Endres and Weber, 1991, IEEE Software 1992).

The idea of a CASE environment was already presented in the early 1970's by (Bubenko et al., 1971). The early CASE environments such as PSL/PSA (Teichroew and Hershey, 1977) were developed in academic research laboratories. PSL/PSA was a powerful environment, but its textual description language hindered its usability at the time when graphical techniques gained

---

[6] McClure (1989) makes a distinction between CASE environment (support for the whole life cycle) and CASE toolkit (support for a part of the life cycle), both of which are included in our CASE environment definition.

popularity (see also Chen, 1988). Rapid advances in data base technology, graphical user interfaces, and storage capacity have made it recently possible to build commercial CASE support for widely used methods based on graphical description languages. Examples of the first popular CASE environments include *Excelerator* (1987) and *Foundation* (1987), while *SelectOMT* is one of the newer products.

One of the main problems in CASE environments is their limited ability to support a method customised to a specific situation. Traditional CASE environments are based on 'built-in' concepts, which form the structure of the repository schema. Description languages may be implemented as a part of the corresponding editor, and they might not be consistently integrated together (Bubenko and Wangler, 1992). This calls for more flexible architectural solutions where conceptual structures and description languages are separated from tool-based information.

The intention of metaCASE technology is to capture the specification of the CASE tool for the required method, and then generate the environment and method support (Alderson, 1991). As commented by Chen (1988) metaCASE tools follow similar *metamodelling* principles (discussed in Section 2.2.1) to those used in compiler-compilers and DSS generators.

In the definition by Bubenko (1988) :

> **Definition 4** A metaCASE environment is a CASE environment including mechanisms to define CASE support for an arbitrary technique or a chain of integrated techniques (a method).

Although hard-coded method support can be more precise than that for modelled methods, the need for metaCASE is recognised. Firstly, there is no CASE support for all methods. A strategy to build tools supporting a local method has been recognised as too costly for organisations (Bubenko, 1988). Secondly, very little is known about the requirements for good design environments (Karrer and Scacchi, 1993), which makes the development difficult. Also, we have presented in Section 1.1.1 the need for analysing and comparing several methods.

Historically, from the late 1970's until the early 1990's, metaCASE environments have been studied and designed mainly in academic research laboratories. Such environments include *SEM* (ISDOS, 1981), *RAMATIC* (Bergsten et al., 1989), *MetaPlex* (Chen and Nunamaker, 1989), *MetaEdit* (Smolander et al., 1991), *MetaView* (Sorenson et al., 1988), and *ConceptBase* (Hahn et al., 1991). During this decade some commercial metaCASE environments have appeared on the market, including *Toolbuilder* by IPSYS (Alderson, 1991), *Maestro II* by Softlab (Merbeth, 1991), *MetaEdit+* by MetaCase Consulting (Kelly et al., 1996), *Paradigm Plus* by Protosoft, and *ObjectMaker* by Mark V Systems.

As our definition above says, most of the metaCASE environments concentrate on the issue of how to define arbitrary modelling techniques. In other words, the focus has been on concept structures and description

languages, whereas process and guidance aspects have not gained such attention. Issues studied include conceptual capabilities (building rich and at the same time versatile metamodelling languages), abilities to define various graphical description languages, and how to provide conceptual and notational flexibility and reuse (see Tolvanen et al., 1996).

In the comparison of three metaCASE environments Marttiin et al. (1993) we studied the following questions: in what detail and how easily can a metaCASE environment define a new technique. The environments were QuickSpec, RAMATIC and Customizer (see Appendix 1). We noticed that RAMATIC is based on a semantically rich but fuzzy meta-metamodel (model of concepts for defining metamodels). Its concepts were semantically overlapping, and thus complex to understand. Customizer was based on a very simple meta-metamodel, although it was mixed with environmental aspects. It can not define methods in such detail as the other two in comparison. QuickSpec was well balanced in terms of richness and simplicity, but it supported textual notations only. The meta-metamodels of these environments are described in Appendix 2. The current situation of metaCASE environments is discussed in (Kelly, 1998).

Support for process or guidance aspects in metaCASE environments has not gained such popularity as conceptual and notational aspects. Conceptual studies that consider process aspects have been done with task structures (Wijers, 1991; Verhoef, 1993), However, some implementations[7] concerning process models exist, including the structured task lists provided in Maestro II, decision based process metamodel *DOT* in ConceptBase (Jarke, 1992), decision based process metamodel implemented in *ALECSI* (Rolland and Cauvet, 1991), MENTOR (SiSaid et al., 1996) and Pro-Art (Pohl, 1996), and the MViews process tool (Grundy and Venable, 1996). Modelling guidance is discussed in connection with the design rationale tool of MetaEdit+ (Kaipala, 1997).

2.1.3 Process Models

Process models have been used since the early days of systems development and software engineering. Models to explicitly show development phases were called life cycle models until the term software process (see, e.g. ICSE, 1987) was introduced and gained popularity. The first phase based models sequenced development into logically ordered tasks (Benington, 1956). Other life cycle models include traditional 'waterfall' models (Royce, 1970; Boehm, 1976), spiral models (Boehm, 1988; Iivari, 1990) and recent object-oriented 'fountain' models (Iivari 1991a; Henderson-Sellers and Edwards, 1990). These models present an ideal structure for systems development process focusing on tasks and products, and the dependencies between these, e.g. sequences, parallelisms, and iterations. These are presented at a coarse granularity level, which has been

---

[7] One should note that we discuss metaCASE environments. There exist many process based CASE environments, such as *Softman* (Mi and Scacchi, 1992). In addition, CASE environments having specific guidance include *HyperCASE* (Cybulski and Reed, 1992) supporting the hypertext navigation between design documents.

criticised as too general to be used in tools and also not describing the elementary development tasks (see e.g. Madhavji, 1991; Curtis et al., 1992).

The first process models were to be the subject of communication between actors involved in a change process contributing to a deeper understanding and learning of the process. These aims have further laid foundations on process improvement and maturity models (Humphrey, 1989; Koch, 1993; Paulk et al., 1993). In process technology there exist multiple objectives for process models, thus introducing a wide range of process modelling approaches. Purposes for process models are discussed in (e.g. ICSE, 1987; Humphrey and Kellner, 1989; Madhavji, 1991; Kellner and Hansen, 1989; Dutton, 1993; Kontio, 1995; Christie, 1995; McChesney, 1995) and the following categories are distinguished: (1) facilitating human understanding and communication, (2) supporting process improvement, (3) supporting project management, and (4) automating guidance and execution (Curtis et al., 1992; Heineman et al., 1994). In recent years, the focus has been on automating processes: many process models are constructed to provide automatic guidance, support, enforcement or execution (McChesney, 1995).

In general, a process is defined as "a set of partially ordered steps intended to reach a goal" (Feiler and Humphrey, 1993; Heineman et al., 1994). We have selected the following definition to separate the model and its language:

> **Definition 5** A process model is a description of a process expressed in a suitable process modelling language (Conradi et al., 1993).

This definition implies that we can (similarly to methods) break up process models into their conceptual structure and description languages (process modelling language; acronym PML). The conceptual structure holds the information to be captured in process models called process features (McChesney, 1995). The process metamodel is a model describing this conceptual structure. It is defined as "a conceptual framework for expressing and composing software process models" (Lonchamp, 1993). PML is a formalism able to represent process models. It adds precise syntax and semantics to process metamodels (Lonchamp, 1993).

McChesney (1995) classifies process features into architectural mechanisms (e.g. hierarchy, scale, parallelism, sequence, constraints), process elements (e.g. agent, activity, artefact, role, and event), and properties (e.g. creation date, duration, and state). The information captured depends on the objectives the model pursues and the perspective which the model adapts (Lonchamp, 1993; Pohl, 1996). Four of the most commonly represented perspectives are functional, behavioural, organisational, and informational perspectives (Curtis et al., 1992). As a result of the multiplicity of process features

> "...a process model is always an abstraction of the reality it represents, and is as such only a partial description, i.e. there are always parts or aspects of the process that are not captured in the model" (Conradi et al., 1993).

The process perspective used and process features selected determine a process paradigm, which is also called a process theory (Rolland et al., 1995). By process paradigm we mean the underlying nature of processes. If we say that processes are interactions between humans, it implies a possibility for uncertain tasks and incomplete process models. In his early discussion Dowson (1987b) distinguished between activity, product, and decision based process paradigms. Another is a division between process, product and agent-based and manufacturing and situational oriented process models (Koskinen, 1996). In spite of the classifications above, we prefer characterising process modelling approaches by their objectives and process metamodels.

We can make a distinction between *descriptive* and *prescriptive* models. After McChesney (1995) descriptive modelling is primarily concerned with explicitly modelling a process currently used in an organisation, whereas the aim in prescriptive modelling is defining the desired or recommended way of executing the process. Descriptive models describe how a system is or has been developed. They can be used to facilitate human understanding, and help in process analysis, assessments and prediction. Examples of descriptive models include an analysis and assessment model TAME (Basili and Rombach, 1988), and traceability models (Pohl, 1996). In contrast, prescriptive models define what can or must be done during the process, thus providing a mechanistic view of the process. McChesney (1995) discusses prescription as being guidance, support, enforcement, or execution of process models. Most process-centred tools are build primarily for prescriptive support.

The level of detail at which process features are captured in models is called the *granularity*. Finding the desired granularity at which to model processes is important in order to gain the benefits from their use. Coarse-grained life cycle models have been said to focus too abstractly on the development process, and to fail to show many elementary process building blocks for managing and co-ordinating the project (Curtis et al., 1992). This is essential for understanding the process, as seen in the case study of modelling software process of the PMR unit in Nokia Telecommunications (Rossi and Sillander, 1997). At the same time, small-grained prescriptive process models are criticised as restricting humans in enacting process (Verhoef and ter Hofstedte, 1995). The level of granularity is connected to another aspect – formality of process models – which is a feature of PMLs.

PMLs are comprehensively discussed in (Curtis et al., 1992, Armenise et al., 1993; Finkelstein et al., 1994; McChesney, 1995). PMLs include logic based rules, attribute grammars, state transition diagrams and Petri nets, Al techniques, and event-trigger based languages among others. The effectiveness of a PML to model a process is dependent on the context in which it is used, the objectives it is used for, and the degree to which its features are understood and used (Armenise et al., 1993). Clearly, we can say that formal PMLs having a strict syntax and semantics are useful for process automation. Humans, however, require understandable PMLs, which do not force one to follow a strictly defined process. As discussed by (Conradi et al., 1993)

> "...interactions among humans and between humans and the tools, that support their activities are characterised by high variability and unpredictability".

Thus, systems design as a multi-person and largely intellectual process can only be partially automated (Armenise et al., 1993).

2.1.4 Process Centred Systems/Software Engineering (PCSE) Environments

The origin of Process Centred Systems/Software Engineering (PCSE) environments is influenced by the adoption of the 'factory' concept to software production in the late 1960s (Cusumano, 1991; Christie, 1995). The initiatives were motivated by a need to reduce cost and improve quality through standardising the way in which software was produced. Another impetus has been the rise of the 'process programming' community (Osterveil, 1987), which resulted in PCSE environments for process automation in the late 1980's and 1990's. The early PCSE environments were based on either process programming or formal rule-based approaches. These include *IStar* (Dowson, 1987a), *Marvel* (Kaiser et al., 1988), and *GRAPPLE* (Huff, 1988).

Fuggetta (1996) summarises functionalities of PCSE environments as follows: they are based on a PML that is used to create process models, they are integrated with facilities to manage and store process artefacts, and the invocation of tools can be directly modelled using the PML. The central role of a process model makes the difference between CASE and PCSE technologies, thus we define a PCSE environment as follows:

> **Definition 6** A Process Centred Systems/Software Engineering (PCSE) environment integrates all elements of a software project explicitly through the medium of the systems development process (Madhavji, 1991).

Several PCSE environments are presented and characterised in (Curtis et al., 1992; Lott, 1993; McChesney, 1995; Finkelstein et al., 1994). PCSE environments can be divided into those integrating development processes with products (reflexive systems) and those including also the concept of user as a part of system definition (workflow and CSCW environments) (Snowdon and Warboys, 1994, p. 3).

PCSE environments are based either on descriptive or prescriptive modelling approaches, but in most cases the approaches are mixed. Descriptive PCSE environments are concerned with the objectives of assessing or predicting processes (see McChesney, 1995). Examples of process assessment include *TAME* (Basili and Rombach, 1988) and the *NATURE* approach[8] (Jarke et al., 1994; Pohl, 1996). The goal of TAME is to collect, validate, and analyse process data, identify problems and make recommendations to improvement. While TAME is focused on later activities of the software process, the NATURE

[8] ESPRIT Basic Research Action 6353, Novel Approaches to Theories Underlying Requirements Engineering.

project addresses requirements engineering tasks: how to capture requirements by using context-oriented process models, and how to manage requirements traceability. Prediction of future process behaviour is supported in *FUNSOFT* nets (Deiters and Gruhn, 1994), among others.

Prescriptive PCSE environments are discussed in the four levels of prescription already mentioned in Section 2.1.3. The first of these levels – automated guidance – may include suggestion of tasks or pending issues, and tool support for it is found for instance in FUNSOFT, *ConversationBuilder* (Kaplan et al., 1992), Marvel, and GRAPPLE. Second, automated support (tool invocations, communication support, workflow support) is found in many PCSE environments, e.g. *Process Weaver* (Fernström et al., 1992), *Appl/A* (Sutton et al., 1990), *MERLIN* (Peuschel et al., 1992), GRAPPLE, and Marvel. Third, automated enforcement means restricting possible process steps according to the current state of enactment, and enforcement functionality is present in ConversationBuilder. Finally, automated execution for well-defined tasks without human involvement is provided in Marvel. More information concerning descriptive and prescriptive PCSE environments is found in (Curtis et al., 1992; McChesney, 1995).

To characterise process modelling environments Dowson (1992) discusses three distinguishable domains: (1) the process modelling domain in which all process artefacts are stored in a process repository, (2) the process enactment domain, which essentially consists of a process engine for enacting defined process models, and (3) the process performance domain, in which the actual process is conducted by humans and tools. In this study we concentrate on the process modelling domain.

2.1.5 Actors and Co-ordination Issues in Systems Development

It is clear that large and complex systems need to be developed in teams. As Winograd and Flores (1986) observed, collaboration and communication exist in all human action except for the simplest tasks. Teamwork also requires tight co-ordination among all efforts involved in the development process. Empirical studies show that projects have difficulties in their group work. For example, communication bottlenecks and breakdowns are very common in software engineering projects (Curtis et al., 1988).

Most systems development methods and process models only give an overview of human interactions. They characterise possible roles for actors to participate in development. Some methods address communication problems far more deeply based on either a political conflict – co-operation game, or a continuous exchange of arguments (Lyytinen, 1987b). Communication difficulties between users and developers have been tackled especially in the *UTOPIA* project (Bødker et al., 1987) and *ETHICS* (Mumford, 1995).

In this study we restrict our interest to formally defined co-ordination. This means that we consider the dependencies between tasks and the ways actors are accessing and manipulating design information, and signalling other actors through models. We do not address direct communication in meetings or messages sent via e-mails. This follows the similar distinction Vessey and

Shravanapudi (1995) make between teamware and groupware environments. Teamware principally co-ordinates activities of group members distinguishing the functions of control, information sharing and monitoring. Groupware, then, is supporting workgroup needs to communicate explicitly about work (e-mail) and schedule their meetings (agendas, calendar management).

When project size and complexity increase, difficulties in co-ordination become more apparent and harder to manage. This may require specific co-ordination technologies to be used (Kraut and Streeter, 1995). Although technical capabilities for co-ordination are available we argue that co-ordination support will long challenge vendors of design environments.

What is the difference between the terms *co-ordination*, *collaboration* and *communication*? Yang (1995) defines these from the software process perspective: co-ordination means ordering of activities in the process, collaboration is management of shared data, and communication is an exchange of information between users. Henderson and Cooprider (1994) include control and co-operative functions in co-ordination, whereas Vessey and Shravanapudi (1995) make a distinction between co-ordination and co-operation technologies putting them into collaboration technology. Furthermore some studies (e.g. Vessey and Shravanapudi, 1995) consider management of shared information as co-ordination.

We define co-ordination with the definition of Malone and Crowston (1994) :

> **Definition 7** Co-ordination is management of dependencies between activities.

Malone and Crowston identify four kinds of dependencies:

- *Management of shared resources.* Whenever multiple activities share some limited resource a resource allocation process is needed to manage the interdependencies among these activities. One important resource in systems development is developers' time.
- *Management of relationships between producers and consumers.* This occurs when one activity produces something that is used by another activity. It requires the sequencing and transfer of products, and ensuring their usability from the perspective of the receiving activities.
- *Simultaneity constraints.* When activities need to occur at the same time activities need to be synchronised.
- *Task/sub-task relationships.* A top-down goal decomposition exists when a group of activities covers all the sub-tasks for achieving some overall goal. When several actors work together to achieve a new goal the process is called bottom-up goal identification.

Malone and Crowston present examples of co-ordination tools for each of the four types above. In our survey (Chapter 4) two process based co-ordination environments ConversationBuilder (Kaplan et al., 1992) and OVAL (Malone et al., 1995) are characterised. Kelly (1998, Chapter 5) discusses the

implementations of co-ordination technology in CASE environments. First of the efforts was the *CoNeX* project (Hahn et al., 1991; Rose et al., 1992), which developed a co-ordination and negotiation tool into *DAIDA* environment. Others include multi-user design environments *TDE* (Taivalsaari and Vaaraniemi, 1997) and MViews (Grundy and Venable, 1996). In the former developers can act synchronously, and the latter provides management for change histories in design models. Co-operation support for PCSE environments is developed in the environments *COO* (Godard et al., 1996) and *SPADE-1* (Bandinelli et al., 1996).

2.1.6 A Contingency View on Systems Development

Development projects face difficult decisions before starting the systems development process. They need to decide the method to be followed, and plan the development process. At the same time, they need to decide what tools to use, and how to use them. What are the factors that drive projects to think about usability of methods and tools instead of developing systems with a similar and standardised manner and why are the right decisions hard to make? An answer for this question can be found from contingency theory.

We can regard both the origin and resulting system (Figure 1.2) as unique artefacts. The change process is dependent on the participating actors. As observed in (Conradi et al., 1993)

> "...processes are human-oriented and the interactions among humans and between humans and the tools, that support their activities are characterised by high variability and unpredictability".

The factors affecting the selection and customisation of methods are discussed in (Davis, 1982; Olle et al., 1991; Euromethod, 1994; van Slooten and Brinkkemper, 1993; Harmsen, 1997). These include economic resources (time and money available), organisation and stakeholder related factors (such as managerial commitments, clarity of goals, importance of system, and knowledge, experience, and resistance of users), and project and user related factors (such as project size, skills, dependencies with other projects, and developers' familiarity with the application area, methods and tools). These facts make systems development situation specific, and explain the need for a critical assessment of suitable methods.

Situations can be characterised using a set of factors called contingency factors or contingencies (Kast and Rosenzweig, 1974; van Slooten and Brinkkemper, 1993). These contingencies are dependent on each other. To study and understand the choices made with respect to methods, we require an understanding of the contingency view. Kast and Rosenzweig (1974) have explained that

> "The contingency view emphasises the multivariable nature of organisations and attempts to understand how organisations operate under varying conditions and in specific circumstances. Contingency views are ultimately directed toward suggesting

> organisational designs and managerial actions most appropriate for specific situations." (Kast and Rosenzweig, 1974, p. 505).

The development situation is not stable. Dependencies between contingencies and values of contingencies evolve. This results in the fact that the optimal selection of a method and its supporting tool is a hard task. As they conclude (Kast and Rosenzweig, 1974, p. 513) "...trying to understand interrelationships and linking the variables together is a painstaking process". The problem of inconclusiveness is present even when we deal with general patterns and rough guidelines such as the selection between prototyping and life cycle models (Burns and Dennis, 1985).

Despite the difficulties in finding out the set of contingency factors influencing the composition of actors, methods, and supporting tools we can say that each development process is unique. Methods as situation based artefacts are discussed in (Sol, 1983; Kumar and Welke, 1992; Avison and Fitzgerald, 1988; Curtis et al., 1988). Their message is that there is no best way to develop systems, and no method is suitable for all development situations. Consequently, in a specific situation all ways to develop a system are not equally efficient.

Contingency based frameworks for systems development are presented in (Olle et al., 1991; Heym and Österle, 1992). In the MultiView method (Avison and Wood-Harper, 1992) a contingency view is applied: the techniques and tools of MultiView are to be used where appropriate, and phases may also be omitted or reduced in scope or executed in a different sequence. Also, a method selection framework is presented (Harmsen, 1997), which is based on the idea that methods are situational and they are better to be composed from method fragments.

Methods have been compared using different frameworks (see, e.g. Olle et al., 1982; Olle et al., 1983, Hackhatron and Karimi, 1988; Song and Osterveil, 1992; Hong et al., 1993; Iivari, 1995). Most of these are feature analyses considering what concepts are covered, what phases are supported, and what is the role of user participation. Although these comparisons are based on limited empirical knowledge of the method use, they show methods' general capabilities and limitations in specific situations. The suitability of methods and associated tools related to specific contingencies is studied in (Naumann and Palvia, 1982; Davis, 1982; Burns and Dennis, 1985; Louadi et al., 1991; van Slooten and Brinkkemper, 1993).

2.1.7 Summary

This section shows our view on systems design as an uncertain and intellectual process. We base our approach on a contingency view. Design methods, process models and supporting tools should be customised for each organisation. In this study we are especially interested in process modelling, for the support of which we are suggesting customised tools. According to the contingency view, both process models and process perspectives are situation specific and they may vary between organisations. This leads to the need for a highly

customisable process modelling approach, in which we are able to customise both process models and process metamodels.

The process modelling tools aim to facilitate production and co-ordination functions in a metaCASE environment. The defined process model is able to manage modelling tasks of any specified method on a granularity level defined in Table 1.2, and to help in co-ordination of design tasks by controlling task assignment, and sequencing, synchronising, and sub-tasking design tasks. In addition, we introduce a language and a user friendly tool set for managing the customisation of process metamodels to determine PMLs. The basis for meta-level customisation is discussed in the next section.

2.2 Method Engineering and Process Engineering

2.2.1 The Use of Metamodelling in Systems Development

The word *meta* originates from the Greek language (*meta* ^) and means higher or beyond. We can use the term *meta-concept* to express things that are on a different (higher) abstraction level than concepts of information systems (Bergheim et al., 1989). For example, the corresponding meta-concepts for the concepts 'data' and 'model' are: *meta-data* meaning 'data of data' and *metamodel* meaning 'a model of a model'. Meta-concept, thus, describes the *type* of the concept, and the concept itself is an *instance* of the type. As an example, Information Resources Dictionary Standard (IRDS) framework (ISO, 1990) defines repository data in four levels (three type-instance pairs). The terms *intension* and *extension* can also be used to denote the terms 'type' and 'instance' (Mark and Roussopoulos, 1986).

In general, metamodelling is "the process of specifying the requirements to be met by the modelling process or establishing the specifications which the modelling process must fulfil" (van Gigch, 1991). In this study, the model of a design method is called a metamodel. Metamodels can be described using a metalanguage or a metalogic. Concerning metamodelling we are interested in the metamodelling language (or logic), metamodelling process and actors participating in metamodelling and developing a metamodel. A metamodel represents the phenomena related to modelling, which correspondingly include the modelling language, modelling process and actors. The modelling language determines the concepts and representations used to describe system models. The levels of metamodelling and modelling are shown in Figure 1.4.

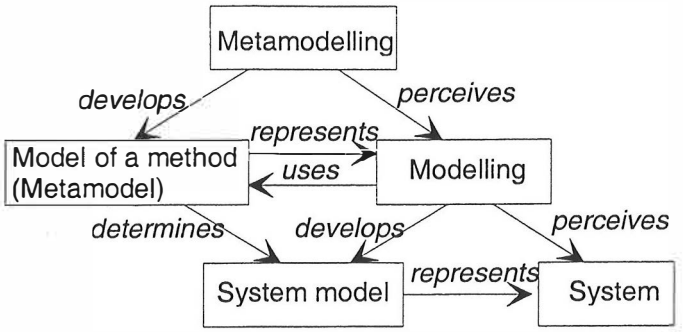

FIGURE 1.4 Metamodelling vs. modelling levels (Tolvanen et al., 1996, revised version of levels in Brinkkemper, 1990)

Method engineering and process engineering (Figure 1.1) are founded on metamodelling. The goal of the method engineering process is to develop a model of a method, and process engineering to develop a process model. More detailed definitions for method and process engineering will be presented in Sections 2.2.2 and 2.2.3. Although we discuss these fields separately these can be seen as different viewpoints on a higher level activity aiming to define a model of systems design including of both method and process model aspects.

Metamodelling can be used for many purposes in ME. Firstly, we may use it for comparing and analysing methods (for examples see Hong et al., 1993; Song and Osterveil, 1992; Rossi and Brinkkemper, 1996; Hillesgersberg, 1997). Metamodelling has been considered an objective way to get information about method similarities and differences. Secondly, we may use it for specifying a new method into a metaCASE environment. In the former case metamodels should be understandable and easily comparable, while in the latter case they need to be strict and may include environment specific constructs (e.g. definition of a CASE editor).

To discuss the PE approaches we need to make a note of the differences in levels. As the process is considered part of modelling, a process model is a part of a metamodel. Furthermore, the process metamodel represents a process modelling language, and belongs on the same level as the meta-metamodel. When we discuss PE we mean engineering on two levels: customising a process model and customising a PML. To avoid confusions we do not use the term *meta-process*, which is used in the PCSE literature to denote all the activities of process (model) evolution/customisation that are accomplished to improve the production process (Fuggetta, 1993; Conradi et al., 1993).

The next sections briefly discuss method engineering and process engineering definitions, approaches, and their tool support. These sections are partially published in Marttiin and Koskinen (1997). We continue the discussion in Section 3, in which we will make a shift from design aid into method engineering aid and discuss method engineering activities by applying a framework by Henderson and Cooprider (1994).

2.2.2 Method Engineering

The effect of metaCASE on method engineering (ME) is to increase the formalisation level of methods used in organisations. It forces organisations to understand, define and use design techniques and methods. At the same time, by allowing incremental modifications in metaCASE, we need to understand the role of ME. ME in relation to metaCASE has concentrated on creating method definitions and translating methods into executable ones. This means the support for modelling concepts and notations. However, in recent years comprehensive support that also focuses on processes, agents, and goals has been included (e.g. Jarke et al., 1998).

The origin of ME according to situational contingencies and accumulated knowledge is described in (Olle et al., 1991; Kumar and Welke, 1992; van Slooten and Brinkkemper, 1993). This moves the subject from conceptual and technical problems of method definition into organisational problems of selecting and composing methods, capturing method knowledge, and learning about method use. ME is also an attempt to provide a framework for the research of methods and supporting tools (Kumar and Welke, 1992; Brinkkemper, 1990). The idea is crystallised in the following definition for ME.

> **Definition 8** Method engineering (ME) is a discipline to design, construct and adapt methods, techniques and tools for systems development. (Brinkkemper, 1996)

The cornerstone when defining a method in a CASE environment is the definition language, i.e. the metamodelling language. A powerful metamodelling language guarantees that a method can be successfully defined. Although metamodelling languages are the most studied aspect of ME, designing and implementing a good one is still one of the main interests. As noted in (Marttiin et al., 1993) we need richer metamodelling languages to be able to capture more details of a method. However, a metamodelling language may become complicated when different method aspects (concepts, notations, agents, tasks, decisions, goals etc.) need to be supported. Metamodelling languages in metaCASE seem to be directed towards focused languages such as GOPRR (Kelly et al., 1996) and PSM (ter Hostedte and van der Weide, 1993) for modelling ontologies and representations, or ETL (Boloix et al., 1991) for consistency rules and transformations, instead of wide and CASE independent ones such as MEL (Harmsen, 1997). Comparisons of metamodelling languages are presented in (Marttiin et al., 1993; Harmsen and Saeki, 1996).

A metamodelling language should not be regarded only as a conceptual artefact. From a method engineer's perspective its usability is related to clear and powerful concepts, but also to the metamodelling notation and support provided by metamodelling tools. In early metaCASE environments method definitions were structured text or rules, which were compiled to executable methods (e.g. Marttiin et al., 1993). During the 1990's more advanced metamodelling tools have been developed, including graphical metamodelling editors in *MetaEdit* (Smolander et al., 1991), textual editors plus graphical

viewers in *Decamerone* (Harmsen, 1997), and form based metamodelling tools in *MetaEdit+* (Kelly et al., 1996).

Another issue relates to building methods from reusable components. For example MetaEdit+ (Kelly et al., 1996) uses abstraction mechanisms such as aggregation (a project as an aggregate of techniques and a technique as an aggregate of modelling types) and specialisation (of modelling types). Such mechanisms also make method integration (i.e. free composition of techniques originally from different methods) easier. Abstraction mechanisms have led to theories of metamodelling hierarchies (Oei and Falkenberg, 1994). The current state of method engineering tools is presented in Harmsen (1997), Kelly (1998) and Rossi (1998).

Since the suggestion of a contingency view for systems development (Davis, 1982) interest in contingency factors (or contingencies) influencing method selection and method suitability has increased. Some methods such as *MultiView* (Avison and Wood-Harper, 1992) contain a framework for selecting techniques appropriate for a project. The interest in contingency factors has led to suggest several classifications (Olle et al., 1991; van Slooten, 1995; Euromethod, 1994; and Harmsen, 1997). Apart from some case studies (van Slooten, 1995) and adaptations in practice (Euromethod, 1994), empirical studies focusing on the impact of contingencies on method definition are lacking.

Some other issues are how to capture knowledge from earlier uses of methods, learning about methods, and how to use the experience gained, which are also essential preconditions for further improvements. The learning aspect has already been tackled in Checkland (1981), influencing other method developers (e.g. Avison and Wood-Harper, 1992). Incremental ME based on gathering knowledge is discussed in Tolvanen and Lyytinen (1993), and learning in relation to a method is described in Mathiassen et al. (1996). In addition, a case study of how to customise OMT to meet the needs of a SE project in Nokia is reported in Aalto (1993). These studies show unique examples of how methods are developed. Because of the limited amount of reports from industry we can not make any generalisation of the ME process.

If the method engineering process is supported by specific computer aided tools, we call the engineering discipline *Computer Aided Method Engineering (CAME)*, and the supporting tools *CAME tools*. We define a CAME environment as follows:

> **Definition 9** A CAME environment is a collection of CAME tools for 1) specifying methods to be used in CASE environments, 2) comparing, analysing, and selecting methods, and/or 3) storing the accumulated knowledge of methods and situational factors (Harmsen et al., 1994a; used in Marttiin et al., 1996).

According to the definition above (the first point) all metaCASE environments contain CAME tools to define methods. In early metaCASE environments tools included text editors and compilers. This is the case with RAMATIC (Bergsten et al., 1989) and also in rule based approaches like ConceptBase (Jarke, 1992).

More advanced CAME support for method definition came with the use of graphical modelling tools with compiling facilities in MetaEdit (Smolander et al., 1991), and form based tools with direct generation of method fragments in MetaEdit+ (Kelly et al., 1996). Tool support for creating method definitions in metaCASE will be discussed more in Section 3.1.1.

Another main line (the second point) is building repository based ME tools supporting selection, composition and, sometimes, analyses of method elements and allowing work separate from the specific metaCASE environment. This includes tool support for task packages (Hidding et al., 1993), method navigation and representation support in Method Base (Saeki et al., 1993), and a prototype for assembling a situational method (Harmsen, 1997). In addition, trials in the use of metrics to analyse methods have been carried out (Rossi and Brinkkemper, 1996). Most of the CAME tools are tackling only the first point of our definition. The examples above show, however, the growing interest for analysing and selecting methods in an unbiased way, and support for composing them easily.

2.2.3 Process Engineering

A new stimulus for process oriented thinking has been triggered by quality approaches such as *Total Quality Management (TQM)* (Deming, 1986), the *ISO 9001* standard (Hall, 1992), and maturity models aiming at process improvement and assessment including the *Capability Maturity Model (CMM)* (Humphrey, 1989; Paulk et al., 1993), *Bootstrap* (Koch, 1993) and *SPICE* (Dorling, 1993). Today many software projects are acquiring productivity increases by continuously improving their processes.

We defined process engineering (PE) earlier as a disciplined approach to manage the creation and customisation of process models. This means that we see the use of process models as a means to process improvement. A process that describes the evolution of a software process is called a meta-process (Fuggetta, 1993; Lonchamp, 1993) or process cycle (Madhavji, 1991). In each evolving process we can distinguish between a production process and process evolution, both of which aim at process improvements.

PE is defined as follows:

> **Definition 10** Process engineering (PE) is a discipline dedicated to study and improvement of software processes. (Lonchamp, 1993).

Kontio (1995) discusses process management[9] as including four activities: process and technology monitoring, process asset management, project management, and process customisation. Process and technology monitoring represent activities that are carried out to accumulate internal experience and analyse externally available technologies. Process asset management

[9] This corresponds to our term PE. Kontio uses the term process engineering in a limited meaning to denote our process customisation.

determines what changes could be made to the process, and project management determines how the process definition is applied in practice. Process customisation/evolution refers to the act of systematically evolving the development process. It stresses only the implementation of the process changes that have been found appropriate for the process.

The idea to support process customisation (as well as other process engineering activities) in PCSE environments has evolved during the 1990s. Conradi et al. (1994) distinguish between template customisation, enactable model customisation (i.e. various project specific models for the same template), and enacting model customisation (i.e. a process model is customised during its execution). In practice systems development projects require enacting model customisation. Because each PCSE environment is based on different architectures, process model evolution is also managed differently in each. Those supporting a meta-process use reflective mechanisms as in *E3* (Baldi et al., 1994) and SPADE, or late binding as in EPOS. Another question is how to find suitable process modelling concepts for users and how to customise process metamodels according to their needs. As we will see in our survey (Chapter 4) this feature is almost ignored in current PCSE environments.

A number of task lists for PE have been presented (Conradi et al., 1993; Christie, 1995; Madhavji, 1991). For example, in Conradi et al. (1993) a structure for PE activities is described including the steps: technology provision, process requirement analysis, process design, process implementation, and process assessment. These activities describe how to carry out process modelling in a PCSE environment. In addition, PE as a group process is tackled in process modelling methods including *Elicit* (Höltje et al., 1994) and *Descriptive Process Modelling (DPM)* (Armitage and Kellner, 1994; Heineman et al., 1994). They provide guidelines on how to carry out the process of capturing process knowledge with the help of and for the use of different stakeholders. These methods are discussed and compared in (Heineman et al., 1994; Kontio, 1995; Rossi and Sillander, 1997). None of the PE methods mentioned above discusses the issue of selecting a process perspective and defining support for it (process metamodel engineering). These studies constrain themselves to the current state of PCSE environments and so do not criticise the fixed process metamodels. We call for more adaptable process perspectives in this study.

Some PE approaches emphasise learning from accumulated experience of processes in the same manner as in ME. These include the TAME project and Experience Factory (e.g. Basili and Rombach, 1987, Basili 1993), and NATURE (e.g. Jarke et al., 1994; Pohl, 1996). The principles in Experience Factory are: separation of responsibilities between product development and process improvement, systematic capture and accumulation of knowledge into Experience Base, continuous learning from experience through measurement, collection and analysis, systematic reuse of accumulated knowledge through packaging and dissemination of knowledge. The NATURE project has similarly focused on experience-based PE. They developed a repository-based environment providing process models for guiding the application engineer, recording traces and relating trace chunks (captured experience) to process definitions. In the case study of software process improvement in a unit of

Nokia Telecommunications (Rossi and Sillander, 1997) one of the issues studied was *PML engineering*. In their study Visual Process Modelling Language (VPML) by Dutton (1993) was customised to correspond to the needs of the software engineering project. A process metamodel was selected and incrementally customised by using interviews and participatory design.

2.2.4 Summary

In this section we have discussed ME and PE processes, method specifications in metaCASE, and process definitions in PCSE environments. Besides this discussion, similarities and differences between ME and PE are presented in (Marttiin and Koskinen, 1998). Metamodelling languages differ according to whether they are aimed at defining methods and process models used in automated environments, or at conceptualising them for understanding, analysis, or comparison.

The MetaEdit+ metaCASE environment allows reuse-based method engineering (Rossi, 1998). This means that we can build meta-data models from components. Our process models are independent from meta-data, so that we are able to attach production actions concerning any type of models, such as OMT Class Diagrams and ER models, into the tasks of process models (see Chapter 6). In addition to the flexible integration between meta-data and process models, this study provides an approach and tools for PML engineering discussed above. We base our approach on the need for customising process metamodels for different process contexts. Process metamodels define the concepts users are working with when enacting a process. In this study we suggest a solution for reuse-based customisation of process metamodels. In addition, the instantiated process models are customisable even when their execution has started. Our prototype environment, CPME, is described in Chapters 5 and 6.

2.3 Related Work

In this section we discuss three disciplines which are close to our approach: *software maturity models*, modelling of workflow processes in *workflow management*, and the use of shared information in *concurrent engineering*.

2.3.1 Software Maturity Models

This study proposes that the suitability of a method and a supporting tool is dependent on the situation. Another perspective can be taken when discussing methods and tools in relation to the maturity of an organisation. In this section we shortly reference some of the well-known maturity models and discuss the relationships between maturity approaches in general and the use of a design environment.

Kontio (1995) points out that the basic philosophy of all maturity models is similar. They are based on Crosby's maturity concept (Crosby, 1979) for assessing the maturity of quality management and are founded on the following principles:

- the product quality is dependent on the process quality,
- the maturity of the process can be characterised by analysing the existence of a set of practices or features, and
- there is an optimal, universal order for implementing these features and compliance to this order determines the maturity of an organisation.

The most widely known and used model is the Capability Maturity Model (Humphrey, 1989; Paulk et al., 1993) developed by the Software Engineering Institute (SEI). The CMM describes 5 levels of maturity (initial, repeatable, defined, measurable, and optimising) which indicate process capability. Each level is characterised by a set of Key Process Areas, which identify principle contributing factors to process capability at each maturity level.

ISO 9001 (ISO, 1987) is a standard for a quality system and states a minimum level of requirements for a system to receive certification. It is a standard motivated to the desire to get a certificate, while the CMM is intended to help in recognising and planning changes. Thus ISO 9001 describes a single level in contrast to the CMM. ISO 9000-3 (ISO, 1991) is a guide on how to apply ISO 9001 for software development.

The Bootstrap maturity model and assessment method (Koch, 1993) is based on CMM and is extended to include many ISO 9000-3 requirements. The Bootstrap method differs from the CMM by assessing the organisation and individual projects separately, by addressing human issues, by allowing a four-scale answer to assessment questions and by using a different calculation algorithm to determine the maturity level.

SPICE (Dorling, 1993) integrates the features and experiences from most of the maturity models. The SPICE practices are divided into *base practices* that contribute to the development of software or management of the development project and *generic practices* that support the implementation and institutionalisation of the process. The SPICE practices cover a larger area of software development than the maturity models presented above. The SPICE approach differs from other assessment methods in that it acknowledges the need to consider an organisation's goals and needs in planning improvements and therefore it is more flexible in use.

Maturity models describe the maturity level of an organisation, but they do not provide much guidance on how to improve processes. The process improvement approaches Experience Factory and NATURE, which are based on learning and collecting experiences, are already discussed in Section 2.2.3.

There exist some assumptions on CASE technology and maturity models. Tate et al. (1992) conclude: "When integrated with metrics and a suitable software process model, CASE can form the testbed for modelling, measurements and management of the processes the fits well with software maturity models". This means that the effective use of such a testbed requires a

certain maturity level, at least the third level of the CMM. We can predict that the adaption of a design environment may be easier in more mature organisations, but also that the use of a design environment may help an organisation to reach a higher maturity level, e.g. with the help of measurements. Sørensen (1993) suggested various arguments for and against CASE adoption at different maturity levels. However, empirical studies on CASE have only looked at correlations to contingencies, e.g. participation, management support, or training have a positive effect on CASE use, not necessarily on the maturity of organisations.

2.3.2 Workflow Management

As we base our work on management of the systems design processes, our approach contains features close to workflow management. *Workflow management* means automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules (Workflow Management Coalition, 1996). Correspondingly, a *Workflow Management System* is a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which are able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications (Workflow Management Coalition, 1996).

Although both workflow management and our process-based design guidance are based on the use of a process model we would like to discuss two elementary differences between these: the need for process evolution and a human as an enactor.

Firstly, workflow management focuses on formal procedures in order to arrive at a decision or a solution. Workflow processes are more predictable and fixed than software processes (Chroust, 1995). In most cases the steps and their sequences in workflow models are defined in advance. Despite industry and office changes, workflows are assumed to be stable unless something changes in the context organisation: labour or tools. We can say that contingencies affecting the changes in process models are more stable in workflow management than in systems development. Such a stability is not possible when we model uncertain and unique processes requiring incremental process evolution.

Secondly, workflow users are seen as enactors: they see themselves as 'officials' following a procedure, making the necessary decisions, based upon facts and judgements (Chroust, 1995). Users are often office people ('end users'), whose routine work the workflow models aim to support. It is assumed that systems development is an intellectual activity that cannot be captured in process models, and developers do not need any process models to be followed. Our approach enables understanding and guidance rather than automating users' tasks.

To conclude, our process approach is aimed at uncertain and intellectual situations, whereas workflow approaches are aimed at modelling stable

processes. Because the customisability requirement differs the workflow modelling approaches are not comparable to ours.

2.3.3 Concurrent Engineering

Concurrent engineering (CE), which is also named Simultaneous Engineering and Collaborative/Co-operative Product Development, aims to increase the concurrency of design by network communication and information sharing in a common repository. It is a systematic approach to the integrated and concurrent design of products and their related processes, including manufacture and support (Winner et al., 1988). CE tools include interoperable methods and tasks, an interoperable computing environment, product data management, design process management, and decision support capabilities (Carter and Baker, 1992). CE serves as an umbrella for integrating various enabling technologies such as Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), CSCW, distributed systems, group decision systems (GDSS), expert systems, multi-media and communication networks (King and Majchrzak, 1996). CASE technology as well as PCSE technology can be located under the same umbrella because they are developing towards groupware environments.

King and Majchrzak (1996) collected, criticised and reformulated assumptions about CE. Some of them – the need for several representations and the role of the process model in design aid – also concern the foundations of this study.

First, an old assumption in CE is that the use of a single representation representing key design features and being understandable by the entire design community is sufficient. However, one representation to show all perspectives of design information is implausible. Furthermore, design information is context specific and the required cognitive model should be the one the community/organisation is accustomed to using. They conclude that "various representations based on explicit cognitive models are needed so that an entire design community can understand key design features vis-à-vis the explicit model".

Another assumption in CE is that "the process of doing design work can be codified resulting in opportunities to intelligently aid design activities". This is close to the view presented by the process programming community (e.g. Osterveil, 1987). The suggested requirement (King and Majchrzak, 1996) that the design process must be thoroughly understood before the process is automated follows the requirement presented in this study. Their conclusions about the difficulties in codifying the patterns of interactions follows the same lines as the findings of modelling patterns in CASE environments (Verhoef and ter Hofstedte, 1995).

3 TOWARDS A UNIFIED VIEW ON CASE AND CAME TECHNOLOGIES

In this section[10] we describe design aids for two levels: systems development using CASE technology and method engineering using CAME technology. The purpose of this chapter is to find out what is the state of the art in CAME technology. More accurately, we link design aid functions to CASE technology and examine what it means if we extend CAME technology into the scope of CASE. The CASE framework describes the scope of method support, which can be defined in CAME. The literature discussed in Section 2.2 is reorganised according to the framework. In the end we discuss the focus of this thesis and each research paper (Chapters 2 to 6) according to the framework.

3.1 A Framework Compounding Design Aid for Systems Development and Method Engineering

A number of CASE technology frameworks have been suggested (Lyytinen et al., 1989; Misra, 1990; Crozier et al., 1989; Wijers, 1991; Heym and Österle, 1992; Fuggetta, 1993; Henderson and Cooprider, 1994). We select a 'functional model' for CASE (Henderson and Cooprider, 1994) which we apply to CAME technology. The selection is based on several reasons. First, the framework takes a comprehensive view of CASE technology. Second, it can be easily adapted to any design aid domain including CAME. The rationale for adapting the framework as such into CAME is found in (Auramäki et al., 1988; Nijssen,

[10] This two level framework is presented in the paper "A Functional Framework for Evaluating Method Engineering Environments: the case of Maestro II/ Decamerone and MetaEdit+." (Marttiin et al., 1996).

1989): according to their architectural principles, CAME technology – a design aid technology for methods and CASE tools – can be treated similarly to CASE technology. Third, the model is based on empirical studies and contains several strictly formulated questions for CASE. This is the main limitation of other framework candidates. Furthermore, our earlier comparison (Marttiin et al., 1993) based on the framework of Lyytinen et al. (1989) did not focus extensively on CAME aspects, and it concentrated on issues not relevant for this study (e.g. portability to use various operating systems, and DBMSs).

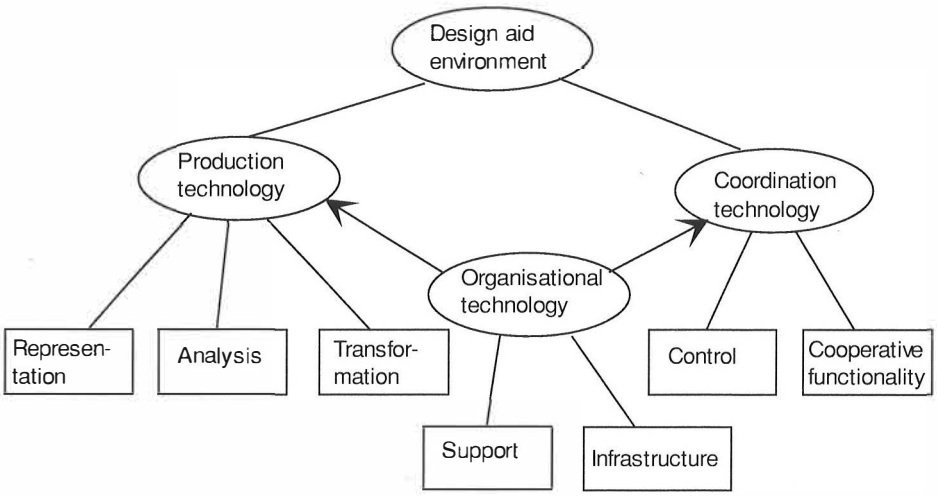

FIGURE 1.5 A functional model of CASE technology (Henderson and Cooprider, 1994)

Henderson and Cooprider (1994) conceptualise design aid technology as a combination of *production*, *co-ordination*, and *organisational* technology (Figure 1.5). Each of these main functions is divided into sub-functions. In the following we present the main issues of each sub-function, briefly discuss the functions for (customised) CASE, and try to capture the basic functions a future CAME environment needs to supply.

3.1.1 Production Technology

Production technology (analysis, design and generation of products from an individual's point of view) is divided into components of *representation*, *analysis* and *transformation* (see Figure 1.5). Representation focuses on abstraction and conceptualisation of phenomena into models. Analysis reflects the problem solving and decision making aspects of development. Transformation calls for rules and mechanisms to transform models.

CASE

Representation in (meta-)CASE calls for the possibility to model systems using various techniques, and to support modelling using alternative notations (textual, diagram, matrix, tabular) (Chen and Nunamaker, 1989). The representation function deals thus with issues of creating, editing, composing, integrating, retrieving, and viewing system models and their components.

Analysis requires verification, validation and simulation support for system models. Verification deals with consistency checking, rules, equivalencies and redundancies in system models, change analysis, and querying system models (Henderson and Cooprider, 1994). Validation can be achieved using metrics, decision aids, requirements tracing, or supporting versions of models. Simulation is used for testing the completeness and performance of a system model by running it.

Transformation considers, for instance, how to transform a logical model into a physical one. The other issues include reverse engineering, change propagation, generation of reports, documentation, and code, and the generation of screen mock-ups and executable code for prototyping.

CAME

While the representation function of CASE focuses on how to model systems, the representation function of CAME focuses on how to specify methods, and how to manage their use in CASE. We consider here the *metamodelling languages*, *abstraction mechanisms* and *notations* used in modelling methods.

- According to earlier studies (Brinkkemper, 1990; Heym and Österle, 1992; Jarke et al., 1994, Marttiin et al., 1995) method fragments (i.e. a method specification or part of it) contain some features of all the CASE framework functions. Thus, in practice the primary focus has been on production: defining a method for representing, analysing, and transforming system models. As noted in our earlier comparison (Marttiin et al., 1993), we need richer metamodelling languages to be able to capture more details of any method aspect. This leads us to consider both the feasibility and the possible granularity of any supported method aspect (Verhoef and ter Hofstedte, 1995).
- In some cases the method fragments are produced from scratch, but the need to store them into a method base, and reuse and recompose them requires the use of abstraction principles. The same basic abstraction principles used in conceptual modelling (Brodie, 1984): *generalisation – specialisation* and *aggregation – decomposition* are suitable for method fragments. Furthermore, the research on metamodelling hierarchies (Oei and Falkenberg, 1994) also introduces *restriction* and *degeneration* principles when considering changes in the metamodelling language, and their effects on modelling languages.

CAME representation deals with editing and viewing method fragments in various notations: structured text, forms, matrices and graphics, for example.

CAME tools have basically supported textual notations as discussed in Section 2.2.2.

Analysis covers *verification, validation* and *simulation* of *method fragments*.

- Verification for methods is mostly situation independent, and thus it can be better formalised and implemented than validation. Verification includes consistency requirements for methods including precedence, input/output, and granularity consistency or the checking of duplicate concepts and uncompleted relationships in methods.
- Kumar and Welke (1992) point out that method engineering follows the incremental learning strategy: every time a project starts the experience and 'wisdom' from earlier successful and unsuccessful projects are evaluated and included into the method fragments. Validation of the method fragments can be performed by comparing them with a number of project factors such as available domain knowledge, the technology used, the organisational size, and the amount of resources available (van Slooten and Brinkkemper, 1993). Some studies (e.g. Euromethod, 1994; Harmsen et al., 1995) offer high-level heuristics to match situations with suitable method fragments.
- A specific simulation can be accomplished when following the 'method engineering by example' strategy (Kelly, 1998), where method components are modelled in the same form as they will appear in CASE.

Transformations in a method base can be divided into *generations between various levels of method fragments, implementation transformations* and *document generation*.

- Transformations between method fragments can be used to partly automate the production of situational methods or to transform a method fragment from a coarse-featured form into a detailed one. Situational methods may require the possibility to combine a set of rules for selecting elementary method fragments and composing them into a rough 'method template'.
- Implementation transformation occurs when we transform a method fragment into an executable form for a CASE environment, or into a form required by another CAME environment. An example of the former is the transformation from *Decamerone* to the *Prolan* language used in *Maestro II* (Harmsen and Brinkkemper, 1995). The bridge between *MetaEdit* and *RAMATIC* (Rossi et al., 1992) serves as an example of the latter case. Transformations between CAME and CASE have mostly been unidirectional. However, to manage system model updates when changing a method may require more seamless solutions to integrate the two levels.
- Creation of method documentation is a CAME representation issue. However, its generation into CASE tool help is a specific transformation issue.

3.1.2 Co-ordination Technology

Co-ordination technology covers functions of *control* and *co-operation*. Henderson and Cooprider (1994) discuss control in terms of *resource management* and *access control*. Resource management enables managers to utilise project resources consistently with project goals. Access control implies ways to manage access rights to user groups that participate in the development. It is closely related to database technology and its mechanisms.

Alternatively, co-operation enables the exchange of information between developers (co-operative modelling) and users (user involvement) for the purpose of influencing the systems development product or process. Henderson and Cooprider (1994) see two kinds of co-operation. The first one is co-operation through design information, such as the ability to attach a note to a diagram. The second one includes co-operative functions that use technology to help facilitate group interactions, such as electronic brainstorming and voting.

CASE

To establish control mechanisms for the CASE environment, we need to deal with, for instance, the concepts of *process*, *project* and *user roles* (Curtis et al., 1992). The development process is both project and method dependent. A process model constructs development tasks in precedence order, integrates techniques and tasks, and allocates users to participate in the tasks. The characterisation of project and user roles is discussed in (Hahn et al., 1991; Curtis et al., 1992; Yu and Mylopoulos, 1994; Marttiin et al., 1995). Other control issues relate to project management, e.g. schedules, deadlines, project complexity metrics, and quality assurance.

Co-operation for CASE calls for the possibility to use co-operative tools, which support messages, notes, anonymous feedback, announcement of changes, and design rationale concerning models or development tasks. We can divide the support into asynchronous communication (electronic mail as an example) and synchronous modelling (several developers editing the same model simultaneously). Other more advanced group interaction mechanisms include brainstorming and other group development techniques that are currently managed using separate tools. The possibility to integrate these into CASE is a challenging future task.

CAME

Method engineering requires the co-ordination and organisational functions, although these do not have such a prominent position as the production function. The current CAME literature does not tackle the co-ordination issues involving multiple method engineers or stakeholders.

We can discuss, however, control and co-operation aspects of method engineering. Controlling method evolution is critical, in particular when the changes to a method take place during the systems development and create effects on CASE and system models. Such effects arise for example when one

deletes a concept in a technique or an attribute of a concept. Co-operation in CAME can be seen as an exchange of method engineering experiences and the use of the collected expertise (Kumar and Welke, 1992). In current visions introduced in the literature, experiences of system designers, software engineers, and project managers is collected, discussed and managed collectively, but transformed into methods by one person, the method engineer.

3.1.3 Support and Infrastructure

All production and co-ordination functions can be supported using organisational technology: the *infrastructure* capturing standard operating procedures and quality standards, and the *support* functionality dealing with organisational guidelines, on-line help, learning aid, 'user friendliness' and 'easiness', which can assist users to understand and use design aid effectively.

CASE

Organisational support for CASE involves help, learning aid and organisational policies for any CASE framework functions: representation, analysis, transformation, control, and co-operation. Firstly, the use of 'additional tools', e.g. hypertext aid, traceability models, process models, browsers, and graphical interfaces can increase the usability of both production and co-ordination functions. These tools can be seen as organisational support when they help an individual user to understand and use CASE effectively. Secondly, method guidance can be produced as a part of the method engineering process and generated to be used in CASE. In this manner the maintenance of method guidance will be easier when methods evolve.

CAME

Similarly to CASE, organisational support for CAME means help, learning aid and organisational policies for the areas of CAME representation, analysis, transformation, control, and co-operation. CAME guidance is not generated from a higher meta-level, rather it collects accumulated experiences of methods use.

The main issues (a bullet for each) and some illustrative examples (italics after the colon) discussed in this section are summarised in Table 1.4.

TABLE 1.4 A functional framework of CASE and CAME

Function	CASE	CAME
Representation	• Modelling of systems using techniques: *concepts, notations, and abstraction mechanisms to represent system models*	• Modelling of methods using a metamodelling language: *concepts, notations and abstraction mechanisms to represent method specifications*
Analysis	• Verification of system models: *consistency checking for models* • Validation of system models: *traceability to requirements, design rationale* • Simulation of system models	• Verification of methods: *consistency checking for method specifications* • Validation of methods: *situational methods, analysis of earlier methods, the use of a method base, ME rationale* • Simulation of methods: *'method engineering by example'*
Transformation	• Transformations between system models • Code and report generation from system models • Generation of screen mock-ups and executable code for prototyping	• Transformations between methods • Generation of methods into CASE: generation of method documentation • Generation of CASE tools & management of repository mappings
Control	• Resource management: *resource capacities, organisational goals, deadlines, priorities, managerial control for CASE* • Access and change control for system models • Process and agent models: *management of design tasks and deliverables, automation of CASE functions*	• Resource management: *resource capacities, organisational goals, deadlines, priorities, managerial control for CAME* • Access and change control for methods: *change effects in CASE* • ME process & agent models: *management of ME tasks and deliverables, automation of CAME functions*
Co-operation	• CASE as co-operative aid: *notes for models, design rationale for system design, co-operative CASE* • Technical support for co-operative CASE: *group interaction mechanisms, asynchronous/ synchronous CASE*	• CAME as co-operative aid: *notes for method specifications, design rationale for ME, co-operative CAME* • Technical support for co-operative CAME: *group interaction mechanisms, asynchronous/ synchronous CAME*
Support & infrastructure	• Help, support, learning aid and organisational policies for CASE representation, analysis, transformation, control, and co-operation: *user friendly CASE tools*	• Help, support, learning aid and organisational policies for CAME representation, analysis, transformation, control, and co-operation: *user friendly CAME tools*

3.2 Summary

Henderson and Cooprider created the framework based on the CASE functions found from the CASE technology of early 1990's. If we use the framework to examine available CAME functionality we can conclude that current CAME technology focuses on production, but does not deal adequately with co-ordination and supporting functions. Furthermore, in production technology there exist many issues that are not sufficiently examined. Examples of the unsolved questions include: how to specify methods with fine granularity, how to reuse method components, and how to propagate method changes to CASE environments.

This study is not an exception among the other CAME studies. We focus on CAME production and are not presenting new solutions for multi-user CAME (e.g. support for co-operative PE). However, one of the main ideas of CPME is to provide a user friendly tool-set for defining, composing, and testing PMLs. Thus, we see our study as focusing also on CAME support.

4 CONCLUSIONS

Various requirements for future CASE are presented in this study (Chapter 2). These requirements cover both single-user and multi-user aspects that belong to the taskware, teamware and groupware functionalities distinguished by Vessey and Shravanapudi (1995). In this study we examine a design environment as a teamware environment. Some authors have studied the possibilities of applying groupware functionality (Rose et al., 1992; Grundy and Venable, 1996; Taivalsaari and Vaaraniemi, 1997). However, at the same time design is seen as the intellectual work of individuals and asynchronous modelling capabilities as a sufficient 'weapon' (Grudin, 1991; Vessey and Shravanapudi, 1995). Empirical studies have mostly been performed with traditional taskware CASE environments. We suggest that researchers focus on empirical studies directed at the use of multi-user environments, in order to gain knowledge of feasible teamware and groupware features in systems design.

A metaCASE environment applying metamodelling to customise methods has proved to be an effective solution. However, in PCSE such a feature has not gained popularity. Current PCSE literature focuses on the question of how to build tools for evolving process models (e.g. Madhavji, 1991; Finkelstein et al., 1994), and how to allow flexibility in PML constructs (e.g. Balzer and Narayanaswamy, 1993; Kaiser et al., 1996). In these studies, however, process metamodels that provide concepts for *interpretation* remain stable. We suppose, however, that such a need to change process modelling concepts is not typical for approaches focused on *machine* interpretation and aimed at automation. We are building support for human understanding and guidance – the purposes of which are not much studied in the technically oriented PCSE environments (McChesney, 1995). This different viewpoint clarifies the limited amount of contributions for process metamodel customisation: providing customisation of the conceptual basis for *human* interpretation. We believe it is essential to provide the concepts, which are understood by designers and which can capture relevant information of the domain. We see the questions of providing

relevant techniques to support systems modelling and providing relevant process perspectives to support the design process to be of the same nature. The customisability features are essential in environments aimed at supporting several different and evolving organisational contexts, and supporting those organisations that otherwise will develop their own in-house process driven design environments. The feature of process metamodel customisability, which we consider important in this study, is adressed only partially in few PCSE environments.

4.1 Summary of the Papers

We discuss here how the paper chapters[11] relate to the CASE and CAME framework discussed in Section 3. The summary is presented in Table 1.5. In Chapter 2 we discuss requirements for CASE, thus covering a wide range of issues, which can be located in each CASE function. Chapter 3 presents a process modelling approach, which aims to both control and guide systems development (CASE control and support). Both of these chapters are based on the use of metamodelling languages, and thus focus on CAME representation. Chapter 4 discusses the need for customising process models and process metamodels, and thus belongs to the level of CAME representation[12]. It also discusses co-ordination support for CASE (CASE co-operation). Chapters 5 and 6 present the CPME prototype, which tackles the question of how to define process models and PMLs (CAME representation). We also discuss consistency checking of PMLs (CAME analysis), and automatic transformations of PMLs to be used in Process Editor (CAME transformation), although these are not the main interests. We consider Process Metamodelling tools as a tool set to define PMLs quickly and easily, thus focusing on CAME support. Our viewpoint in Chapter 6 is to provide organisational support for CASE production and co-ordination (CASE support).

---

[11] The published papers are not in their original forms. We have checked the language and unified the terminology.

[12] Computer Aided Process Engineering (including tools to define and customise process models) is at the level of CAME. Properly, tools to define process metamodels belong one level higher than CAME.

TABLE 1.5 CASE and CAME functions discussed and presented in this study

Function	CASE	CAME
Representation	*Chapter 2*	*Chapters 2, 3, 4, 5 & 6*
Analysis	*Chapter 2*	*(Chapter 5 & 6)*
Transformation	*Chapter 2*	*(Chapter 5 & 6)*
Control	*Chapters 2 & 3*	
Co-operation	*Chapter 2 & 4*	
Support & infrastructure	*Chapters 2, 3 & 6*	*Chapter 5 & 6*

4.1.1 Modelling Requirements for Future CASE: Modelling Issues and Architectural Considerations (Chapter 2)

Research Objectives and Methods

This study discusses ME requirements for future CASE environments. The chosen research approaches, according to Nunamaker's framework, are theory building and system building. An underlying assumption is that a variety of methods needs to be supported (see the discussion of contingencies in Section 2.1.6). The goal of the study is to find out what aspects of systems design can be supported (or even automated) with CASE (theory building). The requirements are classified into three classes based on:

1. requirements for representing conceptual structures and notations related to one technique,
2. requirements related to representing conceptual structures and notations of multiple connected techniques, and
3. requirements related to describing the development process and the participating agents.

We present a model of an architecture for future design environments and examine how the requirements can be captured by using the design architecture (system building).

Results

As a result, we introduce a general model describing an architecture for future CASE environments to integrate method engineering support (support for metamodelling languages) and CASE support. The model covers all three requirement classes above, and it is divided into three abstraction levels: the ISD level, ME level (ISD metalevel), and (ISD) meta-metalevel. The meta-metalevel discusses the concepts for defining methods to determine a metamodelling language. We propose a set of general primitives classified into meta-data types (GOPRR types), activity types and agent types. Finally we

show how most of the requirements can be supported in an environment based on the multi-level model.

This study provided the broad basis for the implementation of the MetaEdit+ environment (Kelly et al., 1996) and GOPRR (Kelly, 1998), although these have changed from our design example.

4.1.2 Towards Flexible Process Support with a MetaCASE Environment (Chapter 3)

Research Objectives and Methods

The goal of this study is to enlarge (meta-)CASE functionality by integrating customisable process modelling support with it. At the time we performed this study, studies on how to provide a conceptual basis (on the meta-metalevel) to define meta-data and process models[13] in an integrated manner had been presented, but implementations were mostly lacking (see Wijers, 1991, p. 9).

The research approach is moving from theory building to system building: the main objective in this study is an initial design for customisable process support. We collected requirements on what kind of process support is needed in metaCASE. From this base we selected human understanding and guidance (instead of automation) to be the main purposes for process support, and graphical process models as means to achieve these purposes.

Results

As a result we design customisable process support. In our design we extend customisability to include two levels — the process metamodel and process model levels. We use Structured Analysis (Yourdon, 1989) as an example method and model a template process model. We examine how to instantiate a process definition into a project specific model and how to make incremental changes on both process metamodel and process model levels.

This study was a precursor of the CPME implementation effort. The first step towards GOPRR-p (process meta-metamodel) is taken, although the metamodelling approach is not yet applied as will be presented in Chapters 5 and 6. Process metamodel customisation, which will be discussed in the survey (Chapter 4), is achieved by specialising primitives (activity types), but leaving the four dependencies (follows, iterates, produces, and uses) fixed. A PML independent Process Editor is presented as an interface to manage production tasks. Furthermore, in our example we discuss the support for incremental customisation of process types and process models.

---

[13] In this study we call process models 'activity models', and process types 'activity types'.

4.1.3 Can Process-Centred Environments Provide The Customised Process Support Needed in MetaCASE? A Literature Review (Chapter 4)

Research Objectives and Methods

This study continues the discussion of what kind of process support is required in metaCASE. We take a closer look at process models which could support co-ordination in a multi-user metaCASE environment. The objective is to examine the suitability of current approaches to provide customised process support for metaCASE. This study presents a framework to discuss and characterise process modelling environments from the perspective of the process approach (design paradigm, visibility, and granularity of process models) and customisability (process models and process metamodels). Ten process centred environments are selected, which are based on the use of process models enactable by humans, and have already existing or designed multi-user support.

The selected research method is a literature survey. This survey examines issues derived from theory building. The framework and findings of this study provide us a tool to compare our environment to the 'state of the art' environments. Therefore, the study is considered observation in Nunamaker's framework.

Results

This study shows the variety of process approaches in the selected process-centred environments. Each environment is based on its own process metamodel. Most of the selected environments contain mechanisms to customise the process metamodel by specialising the primitives (process types), but not by customising the relationships between these primitives. Only OVAL uses metamodelling to define co-ordination oriented tools, and thus is closest to the approach presented in this study. Furthermore, process metamodelling tools as a means to support process metamodel creation and customisation are not presented in any of these environments. Based on this survey we can say that our approach presents a new kind of support for process metamodelling.

4.1.4 Process Support in MetaCASE: Implementing the Conceptual Basis for Enactable Process Models in MetaEdit+ (Chapter 5)

Research Objectives and Methods

The research method used in this chapter is system building. This study discusses the implementation of Process Metamodelling tools. When discussing process metamodels, we locate this study on the meta-metalevel, the highest of the levels presented in Chapters 2 and 3. However, we need to go up one level further, and discuss process metamodelling primitives (see Figure 4.2 in Chapter 4). We replace the earlier approach of specialising process types by a

metamodelling approach. We introduce an implementation of the GOPRR-p process metamodelling language to define process metamodels and a set of Process Metamodelling tools to support the GOPRR-p use.

In addition, we discuss the distinction between user process models to be interpreted by humans and environment process fragments (called actions) to be interpreted by tools. Actions such as 'create a model' are attached to process elements, which appear as 'nodes' that can be enacted by human agents. This distinction gives us a flexible way to manage integration between a variety of process models and a variety of deliverables (system models).

Results

As a result of this study we present the 'higher level' part of the CPME prototype, which allows the customisation of process metamodels. We present the implementation of the GOPRR-p process metamodelling language (including its process engine), and a set of form based Process Metamodelling tools. These tools offer an easy way to define and modify various PMLs to be tested in the Process Editor. A process metamodel determining a PML is created using reusable components, and it is automatically transformed to be used in the Process Editor. For each component we can attach a graphical notation. Integration to system models is achieved via actions, which hold information of tools (to be enacted), products, and operations. Actions are also defined using the Process Metamodelling tools.

CPME also has its limitations. Firstly, metamodelling cannot achieve such exact process behaviour as is provided by the fixed PCSE environments. Furthermore, the current prototype does not contain mechanisms for all complex situations in PMLs. For example, a study in progress concerns improvement of iteration mechanisms for complex situations. Despite these limitations, Process Metamodelling tools serve as a means to create PMLs for guiding the use of a metaCASE environment.

4.1.5 How to Support CASE Activities through Customisable Process Models: Experiments on CPME/MetaEdit+ Using the VPL Formalism and ISPW-6 Example (Chapter 6)

Research Objectives and Methods

The selected research methods in this study are system building and experimentation. The objectives in systems building are twofold: preliminary integration between meta-data, process and agent models, and extending the customisable process modelling environment, CPME, with a Process Editor for defining and enacting process models.

The objectives of the experiment are to study how we can provide support for a new PML, and how MetaEdit+ with CPME works as a teamware environment. We have chosen the Visual Process Language (VPL) (Shephard et al., 1992) to be used as a test PML. The ISPW-6 process example is used to

discuss the multi-user capabilities of MetaEdit+ and the process support provided by CPME.

Results

The results of this study can be divided into system building and experimentation results.

This study presents an integrated view of MetaEdit+'s production, co-ordination and organisational functionality according to the framework by Henderson and Cooprider (see Section 3). We present the production and co-ordination functionality in MetaEdit+ and introduce CPME as a part of organisational functionality that can support these. We have implemented a prototype of Process Editor and a set of agent tools based on a simple agent model. Process Editor can be used for both process modelling and process enactment. The purpose of the latter implementation is to provide task assignment and control for process enactment, thus integrating process and agent models (see Chapter 2).

To discuss CPME experimentation the ISPW-6 example modelled using VPL shows the capabilities of CPME. The result of process metamodelling is that we can model and simulate VPL to a sufficient degree in CPME, although the original tool support for VPL has some differences in relation to MetaEdit+. The main difference is that deliverables are not passing through the process model – only information about their state. In the modelling of the ISPW-6 process we concentrate only on the earlier design stages instead of the software engineering ones also tackled by the ISPW-6 example. The multi-user aspects discussed in the ISPW-6 example (such as access control and sharing of work) are managed using MetaEdit+'s agent models and the multi-user repository. Process and product monitoring, however, are not supported in CPME.

4.2 Limitations of This Study

The limitations of this study can be divided into the limitations of the implementation, and limitations of the research approach according to the research framework by Nunamaker et al. (1991).

Firstly, the process modelling environment presented in this study is part of the MetaPHOR research project. I have participated in the development of MetaEdit and MetaEdit+ since 1989. The development of a real process-centred design environment has not been one of the primary goals of MetaPHOR project, which directed us to take the organisational viewpoint for this study (Henderson and Cooprider, 1994) and to leave some interesting PCSE related questions to be open. For example, our study does not tackle manipulations of model elements. The benefits of context based model manipulation operations are discussed in the joint study by the MetaPHOR project and a research group in RWTH-Aachen (Jarke et al., 1998). Besides, our architecture does not follow the general PCSE environment architecture (Fuggetta, 1996) in which the

process engine has a direct connection to repository data. Our process modelling environment guides the use of tightly coupled CASE tools providing data and control integration (see Kelly, 1998), which leads to the layered architecture presented in Chapter 4 and the separation between meta-engine and process engine presented in Chapter 6. The integration of these engines is possible, but out of the scope in this study.

The CPME implementation contains extensions of existing models and tools in MetaEdit+. The GOPRR-p process metamodelling language is based on the existing GOPRR metamodelling language (Kelly et al., 1996), the process metamodelling tools are based on the existing metamodelling tools (Rossi, 1998), and the Process Editor is based on the existing graphical modelling editor, *Diagram Editor* (MetaEdit+, 1995), which has also been one of my responsibilites in the MetaPHOR project. The reuse of tools and models serves its purpose because it provides an object-oriented way of developing environments. However, in some cases GOPRR solutions aimed at conceptual modelling hampered the implementation of GOPRR-p. One such feature is the question of how to build support for following sequences based on GOPRR's non-directed bindings. To summarise we may ask the question: *Does GOPRR provide a suitable basis for defining process metamodelling languages?* A similar question can be asked when discussing the Process Editor functionality, because it embodies all features of the Diagram Editor.

The questions of to what level of detail and how easily methods can be defined with metaCASE environments yield conflicting answers. We have noticed that complete method support can not be defined by metamodelling. When increasing the level of support we need a richer and more complex metamodelling language and a more detailed and laborious metamodelling process. This is also the case in process models and process modelling languages. The issue of a complete PML specification is related to the purposes of process models: automation requires more strictly defined process models than guidance and human understanding. In CPME our aim is to develop a conceptual basis (GOPRR-p) to support easy definition of various process metamodels and effective customisation of them by using a set of form based Process Metamodelling tools. The VPL example in Chapter 6 shows that we can sufficiently model a process metamodel to be used in MetaEdit+ for our stated purposes. However, this study does not present experiences of defining other process modelling languages.

Secondly, we discuss the limitations of our research approach. One of the most severe limitations in this study is that our environment is not tested in real life, e.g. by case studies on the use of CPME. Real life tests are essential if we want to examine issues such as user-friendliness and understandability in the future design of GOPRR-p and Process Metamodelling tools. Secondly, this study does not discuss organisational adaptation of PCSE environments, although it refers to some studies of CASE adaptation. One of the reasons is that PCSE technology is not used to such a degree as CASE, and there are fewer implementations to test. Because CPME is not tested in real life, it is harder to know what kind of organisations, e.g. in which maturity level an organisation should be, need CPME to support their design process. Our prototype has been

developed and improved driven by conceptual problems – not by organisations' needs. Thirdly, this study does not present other process metamodel examples than VPL. This limits our abilities to present the capabilities and limitations of CPME on a more objective level, because in the example PML selection there always exist some subjective motives.

4.3 Future Work

This study presents an implementation of a customisable process modelling environment (CPME). The implementation is not tested in practice, requiring more observations and experimentations as discussed in Section 4.2. Studies can be divided into those concerning the capabilities of CPME and those concerning its usability. The capability issue focuses on the question of to what level of detail we are able to define various PMLs by using GOPRR-p and the process metamodelling tools. This is discussed in Section 4.3 and requires us to define several PMLs to be used in the Process Editor.

Because the main purposes of our process models are to facilitate human understanding and to guide humans to carry out CASE tasks, the minimum requirement is that the approach is practical and usable. Empirical studies are needed to show the realisation of these aspects. Although CPME can model a PML we need feedback to answer the questions of how easily PMLs can be defined and how meaningful are the GOPRR-p process metatypes. This requires us to use CPME as a part of the MetaEdit+ product in real organisations. System building based on the feedback from the field has been applied in the MetaEdit+ development (Rossi, 1998).

We have only tested the CPME prototype in the Envy/VisualWorks environment without the real Artbase repository. To be able to have a product for organisations we need to make an executable CPME/MetaEdit+ and to test it with the real Artbase repository. We are continuing the development of each CPME tool and are implementing mechanisms to store process models and process metamodels into the Artbase repository. These improvements to the Process Editor include support for checking the process element states due to meta-process activities. We are also starting a project (1) to extend the process metamodelling support with mechanisms for process metamodel constraints, and (2) to design support for reuse based process modelling, including a library of actions to be reused in several process models and developing ways to reuse process templates (see Koskinen, 1997).

4.4 About the Joint Articles

The first article 'Modelling Requirements for Future CASE: Modelling Issues and Architectural Considerations' presents the authors' collective view of metaCASE requirements and a design for a future metaCASE environment. My role as the main author was to collect the outcome of numerous group discussions and seminars. Firstly, I contributed to the design of the metaCASE architecture, especially to the division into three aspects: meta-data, process and agent models. I continued the development of process and agent aspects in the second article (Chapter 3), in which the role of process models to facilitate human understanding and guidance is crystallised. Secondly, I participated in the examination of how the design meets the metamodeling requirements.

My contribution to the fourth article 'Process Support in MetaCASE: Implementing the Conceptual Basis for Enactable Process Models in MetaEdit+' is found in the discussion in Sections 8.1 and 8.2, and in the early version of GOPRR-p to outline the current design. The implementation of GOPRR-p and process metamodelling tools has been done by the first author as her Master's thesis (Koskinen, 1996).

References

Aaen, I., Siltanen, A., Sørensen, C. & Tahvanainen, V.-P. 1992. A Tale of Two Countries - CASE Experiences and Expectations. In: K.E. Kendall, K. Lyytinen, J.I. DeGross (Eds.) The Impact of Computer Supported Technologies on Information Systems Development, Amsterdam: Elsevier Science Publishers, 61-93.

Aalto, J.-M. 1993. Experiences on Applying OMT to Large Scale Systems. In: A. Lehtola, J. Jokiniemi (Eds.) Procs. of the Seminar on Conceptual Modeling and Object-Oriented Programming, Finnish Artificial Intelligence Society, 39-47.

Alderson, A. 1991. Meta-CASE Technology. In: A. Endres, H. Weber (Eds.) Software Development Environments and CASE Technology (LNCS#509), Berlin: Springer-Verlag, 81-91.

Armenise, P., Bandinelli, S., Ghezzi, C. & Morzenti, A. 1993. A survey and assessment of software process representation formalisms. International Journal of Software Engineering And Knowledge Engineering, 3, 3, 410-426.

Armitage, J.W. & Kellner, M.I. 1994. A conceptual schema for process definitions and models. In: Procs. of the 3rd International Conference on the Software Process, IEEE Computer Society Press, 153-165.

Auramäki, E., Leppänen, M. & Savolainen, V. 1988. Universal framework for information systems. Data Base, 19, 1, 11-20.

Avison, D.E. & Fitzgerald G. 1988. Information Systems Development: Methodologies, Techniques and Tools. Oxford: Blackwell.

Avison, D.E. & Wood-Harper, A.T. 1990. Multiview: an exploration in information systems development. Maidenhead: McGraw-Hill.

Baldi M., Gai, S., Jaccheri, M.L. & Lago, P. 1994. E3: Object-Oriented Software Process Model Design. In: A. Finkelstein, J. Kramer, B. Nuseibeh (Eds.) Software Process Modelling and Technology. New York: Wiley, 279-292.

Balzer, R. & Narayanaswamy, K. 1993. Mechanisms for generic process support. In: D. Notkin (Ed.) 1st ACM SIGSOFT Symposium on the Foundations of Software Engineering, Special Issue of Software Engineering Notes, 18, 5, 21-32.

Bandinelli, S., Di Nitto, E. & Fuggetta, A. 1996. Supporting cooperation in the SPADE-1 environment. IEEE Transactions on Software Engineering, 22, 12, 841-865.

Bandinelli, S., Fuggetta, A. & Ghezzi, C. 1993. Software Process Model Evolution in the SPADE Environment. IEEE Transactions on Software Engineering, 19, 12, 1128-1144.

Baskerville, R., Travis, J. & Truex, D. 1992. System Without Methods: The Impact of New Technologies on Information Systems Development Projects. In: K.E. Kendall, K. Lyytinen, J.I. DeGross (Eds.) The Impact of Computer Supported Technologies on Information Systems Development, Amsterdam: Elsevier Science Publishers, 241-269.

Basili, V.R. & Rombach, H.D. 1987. Tailoring the software process to project goals and environments. In: Procs. of the 9th International Conference on Software Engineering, IEEE Computer Society Press, 345-357.

Basili, V.R. & Rombach, H.D. 1988. The TAME Project: Towards Improvement-Oriented Software Environments. IEEE Transactions on Software Engineering, 14, 6, 758-773.

Basili, V.R. 1993. The Experience Factory and its Relationships to Other Improvement Paradigms. In: Procs. of the 4th European Software Engineering Conference, Springer-Verlag.

Benbasat, I. 1984. An analysis of research methodologies. In: W.F. McFarlan (Ed.) The Information Systems Research Challenge, Cambridge: Harward Business School Press, 47-85.

Benington, H.D. 1956. Production of the large computer programs. In: Procs. of the ONR Symposium on Advanced Programming Methods for Digital Computer, 15-27.

Bergheim, G., Sandersen, E. & Solvberg, A. 1989. A taxonomy of concepts for the science of information systems. In: E.D. Falkenberg, P. Lindgreen (Eds.) Information System Concepts: an In-Depth Analysis, Amsterdam: Elsevier Science Publishers, 269-321.

Bergsten, P., Bubenko, J., Dahl, R., Gustafsson, M. & Johansson, L.-Å. 1989. RAMATIC - a CASE shell for implementation of specific CASE tools. SISU, Stockholm, TEMPORA T6.1 report.

Bødker, S., Ehn, P., Kammersgaard, J., Kyng, M. & Sundblad, Y. 1987. An UTOPIAN experience: on design of powerful computer-based tools for skilled graphic workers. In: G. Bjerknes, P. Ehn, M. Kyng (Eds.) Computers and Democracy: A Scandinavian Challenge, Aldershot: Avebury, 251-278.

Boehm, B.W. 1976. Software Engineering. IEEE Transactions on Computers, 25, 12, 1226-1241.

Boehm, B.W. 1987. Improving Software Productivity. IEEE Computer, September, 43-57.

Boehm, B.W. 1988. A spiral model of software development and enhancement. IEEE Computer, 21, 5, 61-72.

Boloix, G., Sorenson, P.G. & Tremblay, J.P. 1991. On Transformations Using A Metasystem Approach To Software Development. The University of Alberta, Edmonton, Technical report.

Brinkkemper, S. 1990. Formalization of Information Systems Modelling. Nijmegen: Thesis Publishers, University of Nijmegen, Ph.D. Thesis.

Brinkkemper, S. 1996. Method engineering: engineering of information systems development methods and tools. Information and Software Technology, 38, 6, 275-280.

Brodie, M. 1984. On the developments of data models. In: M. Brodie, J. Mylopoulos, J. Schmidt (Eds.) Perspectives from Artificial Intelligence, Databases and Programming Languages, Springer-Verlag, 19-47.

Brooks, F.P. Jr. 1975. The Mythical Man-Month. Reading: Addison-Wesley.

Brown, A. 1993. An Examination of the Current State of IPSE Technology, In: Procs. of the 15th International Conference on Software Engineering, IEEE Computer Society Press, 338-347.

Brown, A.W., Earl, A.N. & McDermid, J.A. 1992. Software Engineering Environments: Automated Support for Software Engineering. London: McGraw-Hill.

Bubenko, J.A. jr. & Wangler, B. 1992. Research Directions in Conceptual Specification Development. In: P. Loucopoulos, R. Zicari (Eds.) Conceptual Modeling, Databases and CASE: An Integrated View of Information Systems Development, New York: Wiley.

Bubenko, J.A. jr. 1986. Information System Methodologies - A Research View. In: T.W. Olle, H.G. Sol, A.A. Verrijn-Stuart (Eds.) Information Systems Design Methodologies: Improving the Practice, Amsterdam: Elsevier Science Publishers, 289-318.

Bubenko, J.A. jr. 1988. Selecting a strategy for computer-aided software engineering (CASE). University of Stockholm, SYSLAB Report No 59.

Bubenko, J.A. jr., Langefors, B. & Sølvberg, A. 1971. Computer-Aided Information Systems Analysis and Design. Lund: Studentlitteratur.

Burns, R.N. & Dennis, A.R. 1985. Selecting the Appropriate Development Methodology. Data Base, Fall, 19-23.

Canals, G., Boudjlida, N., Derniame, J.-D., Godard, C. & Lonchamp J. 1994. ALF: A Framework for Building Process-Centred Software Engineering Environments. In: A. Finkelstein, J. Kramer, B. Nuseibeh (Eds.). Software Process Modelling and Technology. New York: Wiley 153-185.

Carter, D.E. & Baker, B.S. 1992. CE (Concurrent Engineering): The product Development Environment for the 1990s. Reading: Addison-Wesley.

Charette, R.N. 1986. Software Engineering Environments: Concepts and Technology. New York: McGraw-Hill.

Checkland, P.B. 1981. Systems Thinking, Systems Practice. Chichester: Wiley.

Chen, M. & Norman, R.J. 1992. A Framework for Integrated CASE. IEEE Software, March, 18-22.

Chen, M. & Nunamaker, J.F. jr. 1989. MetaPlex: an integrated environment for organization and information systems development. In: J.I. DeGross, J.C. Henderson, B.R. Konsynski (Eds.) Procs. of the 10th International Conference on Information Systems, New York: ACM Press, 141-151.

Chen, M. 1988. The Integration of Organization and Information Systems Modeling: A Metasystem Approach to the Generation of Group Decision Support Systems and Computer-Aided Software Engineering. University of Arizona, unpublished Ph.D. Thesis.

Chen, P.P. 1976. The Entity-Relationship Model: Toward a Unified View of Data. ACM Transactions on Database Systems, 1, 1, 9-36.

Chikofsky, E.J. & Rubinstein, B.L. 1988. CASE: reliability engineering for information systems. IEEE Software, March, 11-16.

Christie, A.M. 1995. Software Process Automation: The Technology and Its Adoption. Berlin: Springer-Verlag.

Chroust, G. & Benczur, A. (Eds.) 1994. Workflow Management: Challenges, Paradigms and Products, Oldenbourg, Vienna.

Chroust, G. 1995. Interpretable Process Models for Software Development and Workflow. In: W. Shäfer (Ed.) Procs. of European Workshop on Software Process Technology, EWSPT'95 (LNCS#913), Berlin: Springer-Verlag.

Coad, P. & Yourdon, E. 1991. Object-Oriented Analysis. New Jersey: Prentice-Hall.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. & Jeremes, P. 1994. Object-Oriented Development - The Fusion Method. London: Prentice-Hall.

Conradi, R., Fernström, C. & Fuggetta, A. 1993. A Conceptual Framework for Evolving Software Processes. ACM SIGSOFT Software Engineering Notes, 18, 4, 26-35.

Conradi, R., Larsen, J.-O., Minh, N.N., Munch, B.P. & Westby, P.H. 1994. Integrated Product and Process Management in EPOS. NTH Trondheim, Norway, EPOS Technical Report 222.

Crozier, M., Glass, D., Hughes, J., Johnston, W. & McChesney, I. 1989. Critical analysis of tools for computer-aided software engineering. Information and Software Technology, 31, 9, 486-496.

Crosby, 1979. Quality is Free. New York: McGraw-Hill.

Curtis, B., Kellner, M.I. & Over, J. 1992. Process modeling. Communications of the ACM, 35, 9, 75-90.

Curtis, B., Krasner, H. & Iscoe, N. 1988. A field study of the software design process for large systems. Communications of the ACM, 31, 11, 1268-1287.

Cusumano, M.A. 1991. Japan's Software Factories. New York: Oxford University Press.

Cybulski, J.L. & Reed, K. 1992. A Hypertext Based Software Engineering Environment. IEEE Software, March, 62-68.

Davis, G.B. & Olson, M.H. 1985. Management Information Systems, Conceptual Foundations, Structure and Development. New York: McGraw-Hill.

Davis, G.B. 1982. Strategies for information requirements determination. IBM Systems Journal, 21, 1, 4-30.

Deiters, W. & Gruhn, V. 1994. The Funsoft Net Approach to Software Process Management. International Journal of Software Engineering and Knowledge Engineering, 4, 2, 229-256.

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

Denning, P.J., Comer, D.E., Gries, D., Mulder, M.C., Tucker, A., Turner, A.J. & Young, P.R. 1989. Computing as Discipline. Communications of the ACM, 32, 1, 9-23.

Dorling, A. 1993. SPICE: Software Process Improvement and Capacity dEtermination. Information and Software Technology, 35, 6/7, 404-406.

Dowson, M. 1987a. Integrated Project Support with IStar. IEEE Software, November, 6-15.

Dowson, M. 1987b. Iteration in the software process. In: Procs. of 9th International Conference on Software Engineering, IEEE Computer Society Press, 36-39.

Dowson, M. 1992. Software Process Themes and Issues. In: Procs. of the 2nd International Conference on Software Process, IEEE Computer Society Press, 54-62.

Dutton, J.E. 1993. Commonsense Approach to Process Modeling. IEEE Software, July, 56-64.

Endres, A. & Weber H. (Eds.) 1991. Software Development Environments and CASE Technology. (LNCS#509), Berlin: Springer-Verlag.

Euromethod 1994. Euromethod Architecture, Euromethod project deliverable work package 3.

Excelerator 1987. Excelerator Reference Guide, Index Technology Corporation, Cambridge, USA.

Feiler, P.H. & Humphrey, W.S. 1993. Software Process Development and Enactment: Concepts and Definitions. In: Procs. of the 2nd International Conference on Software Process, IEEE Computer Society Press, 28-39.

Fernström, C., Närfelt, K.-H. & Ohlsson, L. 1992. Software Factory Principles, Architecture, and Experiments. IEEE Software, March, 36-44.

Finkelstein, A., Kramer, J. & Nuseibeh, B. 1994. Software Process Modelling and Technology. New York: Wiley.

Finlay, P.N. & Mitchell, A.C. 1994. Perceptions of the benefits from introduction of CASE: An empirical study. MIS Quarterly, 18, 4, 353-370.

Fitzgerald, B. 1995. The Use of Systems Development Methods: A Survey. University College, Cork, ESRC Research and Discussion Papers, December.

Fitzgerald, B. 1996. Formalized systems development methodologies: A critical perspective. Information Systems Journal, 6, 3-23.

Flood, R.L & Carson, E.R. 1990. Dealing with Complexity: An Introduction to the Theory and Application of Systems Science. New York: Plenum Press.

Forte, G. & Norman, R.J. 1992. A self-assessment by the software engineering community. Communications of the ACM, 35, 4, 28-32.

Foundation 1987. Foundation-Method/1, Information Planning, Version 8.0. Arthur Andersen Consulting, Chicago.

Fuggetta, A. 1993. A Classification of CASE Technology. IEEE Computer, 26, 12.

Fuggetta, A. 1996. Functionality and architecture of PSEEs. Information and Software Technology, 38, 289-293.

Galliers, R.D. 1991. Choosing Appropriate Information Systems Research Approaches: A Revised Taxonomy. In: H.-E. Nissen, H.K. Klein, R. Hirscheim (Eds.) Information Systems Research: Contemporary Approaches and Emergent Traditions. Amsterdam: Elsevier Science Publishers, 327-345.

Garg, P.J. & Jazayeri, M. 1996. Process-Centred Software Engineering Environments. Los Alamitos: IEEE Computer Society Press.

Gane, C. & Sarson, T. 1979. Structured Systems Analysis: Tools and Techniques. New Jersey: Prentice Hall.

van Gigch, J.P. 1991. Systems Design Modeling and Metamodeling. New York: Plenum Press.

Godard, C., Canals, G., Charoy, F., Molli, P. & Skaf, H. 1996. Designing and Implementing COO: Design Process, Architectural Style, Lessons Learned. Procs. of 18th International Conference on Software Engineering, IEEE Computer Society Press, 342-352.

Grudin, J. 1991. Introduction to CSCW. Communications of the ACM, 34, 12, 31-34.

Grundy, J. & Venable, J.R. 1996. Towards an Integrated Environment for Method Engineering. In: S. Brinkkemper, K. Lyytinen, R.J. Welke (Eds.) Method Engineering: Principles of method construction and tool support, London: Chapman & Hall, 45-62.

Hackathorn, R.D. & Karimi, J. 1988. A Framework for Comparing Information Engineering Methods. MIS Quarterly, June, 203-220.

Hahn, U., Jarke, M. & Rose, T. 1991. Teamwork Support in a Knowledge-Based Information Systems Environment. IEEE Transactions on Software Engineering, 17, 5, 467-481.

Hall, T.J. 1992. The quality manual — The applications of BS5750 ISO9001 EN29001. Chichester: Wiley.

Harmsen, F. 1997. Situational Method Engineering. Utrecht: Moret Ernst & Young, Ph.D. Thesis.

Harmsen, F. & Brinkkemper, S. 1995. Design and Implementation of a Method Base Management System for a Situational CASE Environment. In: Procs. of the 2nd Asian-Pacific Software Engineering Conference (APSEC'95), IEEE Computer Society Press, 430-438.

Harmsen, F., Brinkkemper, S. & Oei, H. 1994a. Situational Method Engineering for Information System Projects. In: T.W. Olle, A.A. Verrijn-Stuart (Eds.) Procs. of the IFIP WG8.1 Working Conference CRIS'94, Amsterdam: North-Holland Publishers, 169-194.

Harmsen, F., Brinkkemper, S. & Oei, H. 1994b. A Language and Tool for the Engineering of Situational Methods for Information Systems Development. In: J. Zupancic, S. Wrycza (Eds.) Procs. of the ISD'94 Conference, Bled, Slovenia, 206-214.

Harmsen, F., Lubbers, I. & Wijers, G. 1995. Success-driven Selection of Fragments for Situational Methods: The S3 model. In: K. Pohl, P. Peters (Eds.) Procs. of the 2nd International Workshop on Requirements Engineering: Foundations of Software Quality (REFSQ'95), Aachen: Aachener Beiträge zur Informatik, Band 13, 104-115.

Harmsen, F. & Saeki, M. 1996. Comparison of four method engineering languages. In: S. Brinkkemper, K. Lyytinen, R.J. Welke (Eds.) Method Engineering: Principles of method construction and tool support, London: Chapman & Hall, 209-231.

Heineman, G.T., Botsford, J.E., Caldiera, G., Kaiser, G.E., Kellner, M.I. & Madhavji, N.H. 1994. Emerging technologies that support a software process life cycle. IBM Systems Journal, 33, 3, 501-529.

Henderson, J.C. & Cooprider, J.G. 1994. Dimensions of IS Planning and Design Aids: A Functional Model of CASE Technology. In: T. Allen, M. Scott-Morton (Eds.) IT and the Corporation of the 1990's: Research studies, New York: Oxford University Studies Press, 221-248.

Henderson-Sellers, B. & Edwards, J.M. 1990. The Object-Oriented Systems Life Cycle. Communications of the ACM, 33, 9, 142-159.

Heym, M. & Österle, H. 1992. A reference model of information systems development. In: K.E. Kendall, K. Lyytinen, J.I. DeGross (Eds.) The Impact of Computer Supported Technologies on Information Systems Development, Amsterdam: Elsevier Science Publishers, 215-240.

Hidding, G.J., Freund, G.M. & Joseph, J.K. 1993. Modeling large processes with task packages. Workshop on Modeling in the Large, Washington D.C.: AAAI Conference.

Hillegersberg, J. 1997. Metamodelling-Based Integration of Object-Oriented Systems Development. Amsterdam: Thesis Publishers, University of Rotterdam, Ph.D. Thesis.

Hirscheim, R., Klein, H. & Lyytinen, K. 1995. Information Systems Development - Conceptual and Philosophical Foundations, Cambridge University Press.

ter Hofstede, A.H.M. & van der Weide, Th.P. 1993. Expressiveness in data modeling. Data & Knowledge Engineering, 10, 65-100.

Hong, S., van den Goor, G. & Brinkkemper, S. 1993. A Formal Approach to the Comparison of Object-Oriented Analysis and Design Methodologies. In: J. Nunamaker, R. Sprague (Eds.) Procs. of the 26th Hawaii International Conference on Systems Science, Vol. 4, Los Alamitos: IEEE Computer Society Press, 689-698.

Höltje, D., Madhavji, N.H., Bruckhaus, T. & Hong, W.K. 1994. Eliciting formal models of software engineering processes. In: Procs. of the 1994 CAS conference (CASCON'94), IBM Canada Ltd. and The National Research Council of Canada.

Huff, C.C. 1992. Elements of a Realistic CASE Tool Adoption Budget. Communications of the ACM, 35, 4, 45-53.

Huff, K. 1988. Probing limits to automation: Towards deeper process models. In: C. Tully (Ed.) Procs. of the 4th International Software Process Workshop, IEEE Computer Society Press, 79-81.

Humphrey, W.S. & Kellner, M.I. 1989. Software Process Modeling: Principles of Entity Process Models. In: Procs. of the 11th International Conference on Software Engineering, ACM Press, 331-342.

Humphrey, W.S. 1989. Managing the Software Process. Reading: Addison-Wesley.

ICSE 1987. Procs. of the 9th International Conference on Software Engineering, IEEE Computer Society Press.

IEEE Software 1992. March.

IEEE 1983. IEEE Glossary of Software Engineering Terminology. IEEE Std. 720, New York.

IEEE 1992. Standard for a Software Quality Metrics Methodology. IEEE Std 1061-1992, Institute of Electrical and Electronics Engineers.

Iivari, J. 1990. Hierarchical Spiral Model for Information System and Software Development. Information and Software Technology, 32, 6, 386-399.

Iivari, J. 1991a. Object-oriented Design of Information Systems: The design process. In: F. Van Assche, B. Moulin, C. Rolland (Eds.) Object Oriented Approach in Information Systems, Amsterdam: Elsevier Science Publishers, 61-87.

Iivari, J. 1991b. A paradigmatic analysis of contemporary schools of IS development. European Journal of Information Systems, 1, 4, 249-272.

Iivari, J. 1995. Object-Orientation as structural, functional and behavioural modelling: a comparison of six methods for object-oriented analysis". Information and Software Technology, 37, 3, 155-163.

Iivari, J. 1996. Why are CASE Tools not Used. Communications of the ACM, 39, 10, 94-103.

ISDOS 1981. An Introduction to the System Encyclopedia Manager, The University of Michigan, Ann Arbor, Department of Industrial and Operations Engineering, ISDOS Project Ref #81 SEM-0338-1.

ISO 1987. ISO 9001, Quality systems – Model for quality assurance in design/development, production, installation and servicing, International Statdards Organisation.

ISO 1991. ISO 9000-3, Guidelines for the application of ISO 9001 to the development, supply and maintenance of software, International Statdards Organisation.

ISO/IEC 1990. ISO/IEC 10027 Information Technology - Information Resource Dictionary System (IRDS) - Framework, ISO/IEC International Standard.

Jaccheri, M.L. & Conradi, R. 1993. Techniques for Process Model Evolution in EPOS. IEEE Transactions on Software Engineering, 19, 12, 1145-1156.

Jarke, M. 1992. Strategies for Integrating CASE Environments. IEEE Software, March, 54-61.

Jarke, M., Pohl, K., Rolland, C. & Schmitt, J.-R. 1994. Experience-Based Method Evaluation and Improvement: A process modeling approach. In: T.W. Olle, A.A. Verrijn-Stuart (Eds.) Procs. of the IFIP WG8.1 Working Conference CRIS´94, Amsterdam: North-Holland, 1-27.

Jarke, M., Pohl, K., Weidenhaupt, K., Lyytinen, K., Marttiin, P., Tolvanen, J.-P. & Papazoglou, M. 1998. Metamodelling: A Formal Basis for Interoperability and Adaptability. In: B. Krämer, M. Papazoglou, H.-W. Schmidt (Eds.) Information Systems Interoperability, Advanced Software Development Series, Taunton, Somerset, England: Research Studies Press, 229-263.

Jayaratna, N. 1994. Understanding and Evaluating Methodologies, NIMSAD: A Systemic Framework, Maidenhead: McGraw-Hill.

Joosten, S., Mulder, E. & McCrory, J. 1994. Workflow Literature Guide, version 1. University of Twente, Memoranda Informatica 94-51, September.

Järvinen, P. & Järvinen, A. 1993. Tutkimustyön metodeista. Tietojenkäsittelyopin laitos, Raportti C-1993-2, Tampereen yliopisto.

Kaipala, J. 1997. Augmenting CASE Tools with Hypertext: Desired Functionality and Implementation Issues. In: A. Olivé, J.A. Pastor (Eds.) Advanced Information Systems Engineering (LNCS#1250), Berlin: Springer-Verlag, 217-230.

Kaiser, G.E., Ben-Shaul, I.Z., Popovich, S.S. & Dossick, S.E. 1996. A Metalinguistic Approach to Process Enactment Extensibility. In: Schaefer, W. (Ed.) Procs. of 4th International Conference on the Software Process, IEEE Computer Society Press, 90-101.

Kaiser, G.E., Feiler, P.H. & Popovich, S.S. 1988. Intelligence Assistant for Software Development and Maintenance. IEEE Software, May, 40-49.

Kaiser, G.E., Popovich, S.S. & Ben-Shaul, I.Z. 1993. A Bi-Level Language for Software Process Modeling. In: Procs. of 15th International Conference on Software Engineering, IEEE Computer Society Press, 132-143.

Kaplan, S.M, Tolone, W.J., Carroll, A.M., Bogia, D.P. & Bignoli, C. 1992. Supporting Collaborative Software Development with ConversationBuilder. In: H. Weber (Ed.) Procs. of the 5th ACM SIGSOFT Symposium on Software Development Environments, ACM Press, 11-20.

Karrer, A.S. & Scacchi, W. 1993. Meta-environments for software production. Int. Journal of Software Engineering and Knowledge Engineering, 3, 1, 139-162.

Kast, F.E. & Rosenzweig, J.E. 1974. Organization and Management: A Systems Approach. Second Edition, Tokyo: McGraw-Hill.

Kellner, M.I., Feiler, P.H., Finkelstein, A., Katayama, T., Osterveil, L.J., Penedo, M.H. & Rombach, H.D. 1991. ISPW-6 Software Process Example. Procs. of the 6th International Software Process Workshop, IEEE Computer Society Press. (Available: http://www.idt.unit.no/~epos/ispw6.html).

Kellner, M.I. & Hansen, G.A. 1989. Software Process Modeling: A Case Study. In: Procs. of the 22nd Annual Hawaii International Conference on System Sciences, Vol. II - Software Track, IEEE Computer Society Press, 175-188.

Kelly, S., 1998. Towards a Comprehensive MetaCASE and CAME Environment: Conceptual, Architectural, Functional and Usability Advances in MetaEdit+, Jyväskylä Studies in Computer Science and Information Systems No. 41, University of Jyväskylä, PhD. Thesis.

Kelly, S., Lyytinen, K. & Rossi, M. 1996. METAEDIT+ — A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. In: P. Constantopoulos, J. Mylopoulos, Y. Vassiliou (Eds.) Advanced Information Systems Engineering (LNCS#1080), Berlin: Springer-Verlag, 1-21.

King, N. & Majchrzak, A. 1996. Concurrent Engineering Tools: Are the Human Issues Beign Ignored? IEEE Transactions on Engineering Management, 43, 2, 189-201.

King, S.F. 1996. CASE tools and organisational action, Information Systems Journal, 6, 173-194.

Kirsch, L.J. 1996. The Management of Complex Tasks in Organizations: Controlling the Systems Development Process. Organization Science, 7, 1, 1-21.

Koch, G.R. 1993. Process assessment: The 'BOOTSTRAP' approach. Information and Software Technology, 35, 6/7, 387-403.

Kontio, J. 1995. Promises: A Framework for Utilizing Process Models in Process Asset Management. University of Maryland, Lic. Thesis.

Koskinen, M. & Marttiin, P. 1997. Process Support in MetaCASE: Implementing the Conceptual Basis for Enactable Process Models in MetaEdit+. In: J. Ebert, C. Lewerentz, Software Engineering Environments, IEEE Press, 110-123.

Koskinen, M. 1996. Bringing Process Concepts Alive: on designing process modelling languages in a metaCASE environment. Dept. of Computer Science and Information Systems, University of Jyväskylä, Jyväskylä. (unpublished M.Sc. Thesis)

Koskinen, M. 1997. Beyond Process Modelling Languages: A Metamodelling Approach to Customisable Concepts and Enactability in MetaCASE. In: J.-P. Tolvanen, A. Winter (Eds.) Procs. of 4th Doctoral Consortium on Advanced Information Systems Engineering CAiSE'97, Fachberichte Informatik, Universität Koblenz-Landau.

Kraut, R.E. & Streeter, L.A. 1995. Coordination in Software Development. Communications of the ACM, 38, 3, 69-81.

Kronlöf, K. (Ed.) 1993. Method Integration: Concepts and Case Studies. Wiley Series in Software Based Systems, Chichester: Wiley.

Kronlöf, K., Sheehan, A. & Hallman, M. 1993. The Concept of Method Integration. In: K. Kronlöf, (Ed.) Method Integration: Concepts and Case Studies. Wiley Series in Software Based Systems, Chichester: Wiley, 1-18.

Kumar, K. & Welke, R. J. 1992. Methodology Engineering®: A Proposal for Situation-specific Methodology Engineering. In: W.W. Cotterman, J.A. Senn (Eds.) Challenges and Strategies for Research in Systems Development, Chichester: Wiley, 257-269.

Land, F.F. & Kennedy-McGregor, M. 1987. Information and Information Systems: Concepts and Perspectives. In: R. Galliers (Ed.) Information Analysis: Selected Readings, Sydney: Addison-Wesley, 63-91.

Lehman, M.M. 1987. Process Models, Process Programs, Programming Support. In: Procs. of the 9th International Conference on Software Engineering, IEEE Computer Society Press, 14-16.

Lehman, M. & Turski, W. 1987. Essential properties of IPSEs. ACM SIGSOFT Software Engineering Notes, 12, 1, 52-56.

LeQuesne, P.N. 1990. Individual and Organisational Factors in the Design of Integrated Project Support Environments. London Business School, Ph.D. Thesis.

Lonchamp, J. 1993. A structured conceptual and terminological framework for software process engineering. In: Procs. of the 2nd International Conference of Software Process, IEEE Computer Society Press, 41-53.

Lonchamp, J. 1994. An Assessment Exercise. In: A. Finkelstein, J. Kramer, B. Nuseibeh (Eds.). Software Process Modelling and Technology. New York: Wiley, 335-356.

Lott, C.M. 1993. Process and Measurement Support in SEEs. ACM SIGSOFT Software Engineering Notes, 18, 4, 83-93.

Louadi, M.E., Pollalis, Y.A. & Teng, J.T.C. 1991. Selecting a Systems Development Methodology: A Contigency Framework. Information Resources Management Journal, 4, 1, Winter.

Lundeberg, M., Goldkuhl, G. & Nilsson, A. 1980. Information systems development - a systematic approach. New Jersey: Prentice-Hall.

Lyytinen, K. 1987a. A Taxonomic Perspective of Information Systems Development: Theoretical Constructs and Recommendations. In: R.J. Boland, R.A. Hirschheim (Eds.) Critical Issues in Information Systems Research, Chichester: Wiley, 3-42.

Lyytinen, K. 1987b. Different Perspectives on Information Systems: Problems and Solutions. ACM Computing Surveys, 19, 1, 5-46.

Lyytinen, K., Kaipala, J., Kelly, S., Koskinen, M., Liu, H., Luoma, J., Marttiin, P., Pohjonen, R., Oinas-Kukkonen, H., Rossi, M., Somppi, M., Tahvanainen, V.-P. & Tolvanen, J.-P. 1997. CAMSO Computer Aided Modelling Support for Organisations. Technical Report TR-18, Computer Science and Information Systems Reports, University of Jyväskylä.

Lyytinen, K., Kerola, P., Kaipala, J., Kelly, S., Lehto, J., Liu, H., Marttiin, P., Oinas-Kukkonen, H., Pirhonen, J., Rossi, M., Smolander, K., Tahvanainen, V.-P. & Tolvanen, J.-P. 1994. MetaPHOR: Metamodeling, Principles, Hypertext, Objects and Repositories, Technical Report TR-7, Computer Science and Information Systems Reports, University of Jyväskylä.

Lyytinen, K., Smolander, K. & Tahvanainen, V.-P. 1989. Modelling CASE Environments in Systems Work. CASE89 conference papers, Kista, Sweden.

Madhavji, N.H. 1991. The process cycle. Software Engineering Journal, 6, 5, 234-242.

Malone, T.W. & Crowston, K. 1994. The interdisciplinary study of coordination. ACM Computing Surveys, 26, 1, 87-119.

Malone, T.W., Lai, K.-Y. & Fry, C. 1995. Experiments with Oval: A Radically Tailorable Tool for Cooperative Work. ACM Transactions on Information Systems, 13, 2, 177-205.

Mark, L. & Roussopoulos, N. 1986. Metadata management. IEEE Computer, 19, 12, 26-36.

Marttiin, P. 1991. Metamodeling in CASE environments (in Finnish Metamallintaminen CASE-ympäristöissä). Tutkimuksia TU-9, Department of Computer Science and Information Systems, University of Jyväskylä.

Marttiin, P. 1994. Towards Flexible Process Support with a CASE Shell. In: G. Wijers, S. Brinkkemper, T. Wasserman (Eds.) Advanced Information Systems Engineering (LNCS #811), Berlin: Springer-Verlag, 14-27.

Marttiin, P. 1997. Can process-centred environments provide the customised process support needed in metaCASE? a literature review. Many Facets of Process Engineering Workshop, pp. 165-180.

Marttiin, P. 1998. How to support CASE activities through fully customisable process models: Experiments of CPME/MetaEdit+ using VPL formalism and ISPW-6 example, to be published at ICSP conference, Chicago, USA.

Marttiin, P., Harmsen, F. & Rossi, M. 1996. A Functional Framework for Evaluating Method Engineering Environments: the case of Maestro II/ Decamerone and MetaEdit+. In: S. Brinkkemper, K. Lyytinen, R.J. Welke (Eds.) Method Engineering: Principles of method construction and tool support, London: Chapman & Hall, 63-86.

Marttiin, P. & Koskinen, M. 1998. Similarities and differences of method engineering and process engineering approaches, to be published at IRMA conference, Boston, USA.

Marttiin, P., Lyytinen, K., Rossi M., Tahvanainen V.-P., Smolander K. & Tolvanen, J.-P. 1995. Modeling Requirements for Future CASE: modeling issues and architectural considerations. Information Resource Management Journal, 8, 1, 15-25.

Marttiin, P., Lyytinen, K., Rossi, M., Smolander, K, Tahvanainen, V.-P. & Tolvanen, J.-P. 1992. Modeling requirements for future CASE: issues and implementation considerations. In: J.I. DeGross, J.D. Becker, J.J. Elam (Eds.) Procs. of the 13th International Conference on Information Systems, ACM Press, 9-20.

Marttiin, P., Rossi, M., Tahvanainen, V.-P. & Lyytinen, K. 1993. A comparative review of CASE Shells: a preliminary framework and research outcomes. Information and Management, 25, 11-31.

Mathiassen, L., Munk-Madsen, A., Nielsen, P.A. & Stage, J. 1996. Method engineering: Who's the Customer. In: S. Brinkkemper, K. Lyytinen, R.J. Welke (Eds.) Method Engineering: Principles of method construction and tool support, London: Chapman & Hall, 233-245.

McChesney, I.R. 1995. Toward a classification scheme for software process modeling approaches. Information and Software Technology, 37, 7, 363-374.

McClure, C. 1989. CASE is Software Automation. New Jersey: Prentice-Hall.

MetaEdit+ 1995. MetaEdit+: Method Workbench User's Guide (version 2.0), MetaCase Consulting, Jyväskylä: MicroWorks Finland.

Merbeth, G. 1991. Maestro II - das integrierte CASE-System von Softlab. In: H. Balzer (Ed.) CASE Systeme und Verzeuge, BI Wissenschaftsverlag, 319-336.

Merbeth, G. 1997. Software Engineering Environments: an Industrial Perspective. Keynote Speech in Software Engineering Environments, Gottbus, Germany.

Mi, P. & Scacchi, W. 1992. Process Integration in CASE Environments. IEEE Software, March, 45-53.

Misra, S. 1990. CASE system characteristics: evaluative framework. Information and Software Technology, 32, 6, 415-422.

Mumford, E. 1995. Effective Requirements Analysis and Systems Design: The ETHICS Method. Basingstoke: Macmillan.

Naumann, J.D. & Palvia, S. 1982. A Selection Model for System Development Tools. MIS Quarterly, March, 39-48.

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T. & Swartout, W.R. 1991. Enabling Technology for Knowledge Sharing. AI Magazine, Fall, 36-55.

Nijssen, G.M. 1989. An Axiom and Architecture for Information Systems. In: E.D. Falkenberg (Ed.) Information System as an In-Depth Analysis, Amsterdam: Elsevier Science Publishers, 157-175.

Norman, R. & Nunamaker, J.F. jr. 1989. CASE productivity perspections of software engineering professionals. Communications of the ACM, 32, 9, 1102-1108.

Norman, R.J., Stevens, W., Chikofsky, E.J., Jenkins, J., Rubinstein, B.L. & Forte, G. 1991. CASE at the Start of 1990's. In: Procs. of 13th International Conference on Software Engineering, IEEE Computer Society Press, 128-139.

Nunamaker, J.F. jr. 1992. Build and Learn, Evaluate and Learn. Informatica — The Journal of Management Information Systems Development, 1, 1, 1-6.

Nunamaker, J.F. jr., Chen, M. & Purdin, T.D.M. 1991. Systems development in Information Systems Research. Journal of Management Information Systems, 7, 3, 89-106.

Oei, J.L.H. & Falkenberg, E.D. 1994. Harmonisation of Information System Modelling and Specification Techniques. In: T.W. Olle, A.A. Verrijn-Stuart (Eds.) Procs. of the IFIP WG8.1 Working Conference CRIS'94 Amsterdam: Elsevier Science Publishers, 151-168.

Olerup, A. 1991. Design approaches: a comparative study of information system design and architectural design. The Computer Journal, 34, 215-224.

Olle, T.W., Hagelstein, J., MacDonald, I.G., Rolland, C., Sol, H.G., Van Assche, F.J.M. & Verrijn-Stuart, A.A. 1991. Information Systems Methodologies — A framework for understanding. Wokingham: Addison-Wesley.

Olle, T.W., Sol, H.G. & Tully, C.J. (Eds.) 1983. Information Systems Design Methodologies: A Feature Analysis. Amsterdam: Elsevier Science Publishers.

Olle, T.W., Sol, H.G. & Verrijn-Stuart, A.A. (Eds.) 1982. Information Systems Design Methodologies: A comparative review. Amsterdam: Elsevier Science Publishers.

Olle, T.W., Sol, H.G. & Verrijn-Stuart, A.A. (Eds.) 1986. Information Systems Design Methodologies: Improving the Practice. Amsterdam: Elsevier Science Publishers.

Orlikowski, W. 1988. CASE Tools and the IS Workplace: Some Findings from Empirical Research. In: Procs. of the 1988 ACM SIGCPR Conference on The Management of Information Systems Personnel, ACM Press, 88-97.

Osterweil, L.J. 1987. Software processes are software too. Procs. of the 9th International Conference on Software Engineering, IEEE Computer Society Press, 12-13.

Paulk, M.C., Curtis, B., Chrissis, M.B. & Weber, C.V. 1993. The Capability Maturity Model: Version 1.1. IEEE Software, July, 18-27.

Peuschel, B., Schäfer, W. & Wolf, S. 1992. A Knowledge-Based Software Development Environment Supporting Cooperative Work. Int. Journal of Software Engineering & Knowledge Engineering, 2, 1, 79-106.

Pohl, K. 1996. Process-Centered Requirements Engineering. New York: Wiley.

Rational Software Corporation 1997. UML Document Set. Available: http://www.rational.com/uml/references/index.

Rolland, C. & Cauvet, C. 1991. ALECSI - An Expert System for Requirements Engineering. In: R. Andersen, J. Bubenko, A. Sølvberg (Eds.) Advanced Information Systems Engineering (LNCS#498), Berlin: Springer-Verlag, 31-49.

Rolland, C., Souveyet, C. & Moreno, M. 1995. An approach of defining ways-of-working. Information Systems, 20, 4, 337-359.

Rose, T., Maltzahn, C. & Jarke, M. 1992. Integrating Object and Agent Worlds. In: P. Loucopuolos (Ed.) Advanced Information Systems Engineering, Berlin: Springer-Verlag, 17-32.

Rossi, M. 1998. Working towards comprehensive computer support for Method Engineering: Implementation of CAME Environment in MetaEdit+, Jyväskylä Studies in Computer Science and Information Systems No. 42, University of Jyväskylä, Ph.D. Thesis.

Rossi, M. & Brinkkemper, S. 1996. Complexity Metrics for Systems develeopment Methods and Techniques. Information Systems, 21, 2, 209-227.

Rossi, M. 1993. Transformations in CASE Shell Environments. Department of Computer Science and Information Systems, University of Jyväskylä, Working paper WP-18, Lic. Thesis.

Rossi, M., Gustafsson, M., Smolander, K., Johansson, L.-Å. & Lyytinen, K. 1992. Metamodeling Editor as a Front End Tool for a CASE Shell. In: P. Loucopuolos (Ed.) Advanced Information Systems Engineering (LNCS#593), Berlin: Springer-Verlag, 546-567.

Rossi, S. & Sillander, T. 1997. Four Fundamental Software Process Modelling Principles: The Case of Nokia Telecommunications. Dept. of Computer Science and Information Systems, University of Jyväskylä. (unpublished M.Sc. Thesis)

Royce, W.W. 1970. Managing the Development of Large Software Systems. In: Procs. Wescon, New York: IEEE Press, 1-9. (Reprinted in Procs. of the 9th International Conference on Software Engineering, IEEE Computer Society Press, 1987, 328-338)

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. & Lorensen, W. 1991. Object-Oriented Modeling and Design. New Jersey: Prentice-Hall.

Russo, N., Wynekoop, J. & Waltz, D. 1994. The use and adaptation of system development methodologies. In: Procs. of International Conference of IRMA, Atlanta.

Saeki, M., Iguchi, K., Wen-yin, K. & Shinohara, M. 1993. A meta-model for representing software specification & design methods. In: N. Prakash, C. Rolland, P. Pernici (Eds.) Procs. of the IFIP WG8.1 Conference on Information Systems Development Process, Como, pp. 149-166.

Scheer, A. 1994. Business Process Engineering: Reference Models for Industrial Enterprises, Berlin: Springer-Verlag.

Shephard, T., Sibbald, S. & Wortley, C. 1992. A Visual Software Process Language. Communications of the ACM, 35, 4, 37-44.

Si-Said, S., Rolland, C. & Grosz, G. 1996. MENTOR: A Computer Aided Requirements Engineering Environment. In: P. Constantopoulos, J. Mylopoulos, Y. Vassiliou (Eds.) Advanced Information Systems Engineering (LNCS#1080), Berlin: Springer-Verlag, 22-43.

van Slooten, K. & Brinkkemper, S. 1993. A Method Engineering Approach to Information Systems Development. In: N. Prakash, C. Rolland, B. Pernici (Eds.), Information Systems Development Process, Amsterdam: Elsevier Science Publishers, 167-186.

van Slooten, K. 1995. Situated Methods for Systems Development. Den Haag: Cip-Data Koninklijke Bibliotheek, Ph.D. Thesis.

Smolander, K. 1992. OPRR - A Model for Methodology Modeling. In K. Lyytinen, V.-P. Tahvanainen (Eds.) Next Generation of CASE Tools, Studies in Computer and Communication Systems, Amsterdam: IOS press, 224-239.

Smolander, K., Lyytinen, K., Tahvanainen, V.-P. & Marttiin, P. 1991. MetaEdit - A flexible graphical environment for methodology modelling. In: R. Andersen, J. Bubenko, A. Sølvberg (Eds.) Advanced Information Systems Engineering (LNCS#498), Berlin: Springer-Verlag, 168-193.

Snowdon, R.A. & Warboys, B.C. 1994. An Introduction to Process-Centred Environments. In: A. Finkelstein, J. Kramer, B. Nuseibeh (Eds.), Software Process Modelling and Technology, New York: Wiley, 1-8.

Sol, H.G. 1983. A Feature Analysis of Information Systems Design Methodologies: Methodological Considerations. In: T.W. Olle, H.G. Sol, C.J. Tully (Eds.) Information Systems Design Methodologies: A Feature Analysis, Amsterdam: Elsevier Science Publishers, 1-7.

Song, X. & Osterveil, L. 1992. Towards the Objective and Systematic Comparisons of Software Design Methodologies. IEEE Software, 18, 5, 43-53.

Sørensen, C. 1993. Introducing CASE Tools into Software Organizations. Dept. of Mathematics and Computer Science, Institute of Electronic Systems, Aalborg University. Ph.D. Thesis.

Spivey, J.M. 1988. Understanding Z, a specification language and its formal semantics. Cambridge Tracts in Theoretical Computer Science, nr 3.

Sorenson, P.G., Tremblay, J-P. & McAllister, A.J. 1988. The Metaview system for many specification environments. IEEE Software, 30, 3, 30-38.

Sutton, S.M., Heimbigner, F.D. & Osterveil, L.J. 1990. Language constructs for managing change in process centered environments. ACM SIGSOFT Software Engineering Notes, 15, 6, 206-217.

Taivalsaari, A. & Vaaraniemi, S. 1997. TDE: Supporting Geographically Distributed Software Design with Shared, Collaborative Workspaces. In: A. Olivé, J.A. Pastor (Eds.) Advanced Information Systems Engineering, LNCS#1250, Berlin: Springer-Verlag, 81-87.

Tate, G., Verner, J. & Jeffery, R. 1992. CASE: A Testbed for Modeling, Measurement and Management. Communications of the ACM, 35, 4, 65-72.

Teichroew, D. & Hershey III, E.A. 1977. PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems. IEEE Transactions on Software Engineering, 3, 1, 41-48.

Terry, B. & Logee, D. 1990. Terminology for software engineering environment (SEE) and computer-aided software engineering (CASE). ACM SIGSOFT Software Engineering Notes, 15, 2, 83-94.

Tolvanen, J.-P. & Lyytinen, K. 1993. Flexible method adaptation in CASE - the metamodeling approach. Scandinavian Journal of Information Systems, 5, 51-77.

Tolvanen, J.-P., Marttiin, P. & Smolander, K. 1993. An Integrated Model for Information Systems Modeling. In: J. F. Nunamaker, R.H. Sprague (Eds.) Procs. of the 26th Annual Hawaii International Conference on Information System Science. Vol 3., IEEE Computer Society Press, 470-479.

Tolvanen J.-P., Rossi M. & Marttiin P. 1992. Comparison of Three Object-Oriented Methods; a Method Adaptation Perspective. In: M. Rossi (Ed.) CASE for Object-Oriented Methods — Position papers of the ECOOP'92 WorkShop W3, Working Paper WP-22, Department of Computer Science and Information Systems, University of Jyväskylä, Finland, pp. 38-43.

Tolvanen, J.-P., Rossi, M. & Liu, H. 1996. Method Engineering: Current research directions and implications for future research. In: S. Brinkkemper, K. Lyytinen, R.J. Welke (Eds.) Method Engineering: Principles of method construction and tool support, London: Chapman & Hall, 296-317.

Turner, J.A. 1987. Understanding the Elements of System Design. In: R.J. Boland jr., R.A. Hirschheim (Eds.) Critical Issues in Information Systems Research, Chichester: Wiley, 97-112.

Verhoef, T.F. & ter Hofstedte, A.H.M. 1995. Feasibility of Flexible Information Modelling Support. In: J. Iivari, K. Lyytinen, M. Rossi (Eds.) Advanced Information Systems Engineering (LNCS#932), Berlin: Springer-Verlag, 168-185.

Verhoef, T.F. 1993. Effective Information Modelling Support. Den Haag: Cip-Data Koninklijke Bibliotheek, Ph.D. Thesis.Vessey, I. & Sravanapudi, A.P. 1995. CASE Tools as Collaborative Support Technologies. Communications of the ACM, 38, 1, 83-94.

Vessey, I., Jarvenpaa, S. & Tractinsky, N. 1992. Evaluation of Vendor Products: CASE Tools as Methodology Companions. Communications of the ACM, 35, 4, 90-105.

Vessey, I. & Sravanapudi, A.P. 1995. CASE Tools as Collaborative Support Technologies. Communications of the ACM, 38, 1, 83-94.

Wand, Y. 1996. Ontology as a foundation for meta-modelling and method engineering. Information and Software Technology, 38, 281-287.

Wasserman, A.I., Freeman, P. & Porcella, M. 1983. Characteristics of Software Development Methodologies. In: T.W. Olle, H.G. Sol, C.J. Tully (Eds.) Information Systems Design Methodologies: A Feature Analysis, Amsterdam: Elsevier Science Publishers, 37-62.

Welke, R. 1983. IS/DSS: DBMS support for information systems development. In: G.W. Holsapple, A.B. Whinston (Eds.) Data Base Management: Theory and Applications, Dortrecht: Reidel Publishing Company, 195-250.

Wijers, G. & van Dort, H. 1990. Experiences with the use of CASE tools in the Netherlands. In: B. Steinholz, A. Sølvberg, B. Bergman (Eds.) Advanced Information Systems Engineering (LNCS#436), Berlin: Springer-Verlag, 5-20.

Wijers, G. 1991. Modeling Support in Information Systems Development, Thesis publishers, Amsterdam, Ph.D. Thesis.

Winner, R.I., Pennell, J.P., Bertrand, H.E. & Slusarezuk, M.M.G. 1988. The Role of Concurrent Engineering in Weapons System Acquisition, IDA Report R-338, Institute for Defense Analyses, Alexandria VA.

Winograd, T. & Flores, F. 1986. Understanding computers and cognition. Norwood: Ablex.

Workflow Management Coalition. 1996. The Workflow Management Coalition Specification, Terminology & Glossary, Document Number WFMC-TC-1011, Document Status - Issue 2.0.

Wynekoop, J.D. & Conger, S.A. 1991. A review of computer aided software engineering research methods. In: H.-E. Nissen, H.K. Klein, R. Hirschheim (Eds.) Information Systems Research: Contemporary Approaches and Emergent Traditions, Amsterdam: Elsevier Science Publishers, 301-320.

Yang, Y. 1995. Coordination for process support is not enough. In: W Shäfer (Ed.) Procs of European Workshop on Software Process Technology, EWSPT'95 (LNCS#913), Berlin: Springer-Verlag, 205-208.

Yourdon, E. 1989. Modern Structured Analysis. London: Prentice-Hall.

Yourdon, E. 1992. The Decline and Fall of the American Programmer. New Jersey: Prentice-Hall.

Yu, E.S.K. & Mylopoulos, J. 1994. Understanding "Why" in Software Process Modelling, Analysis, and Design. In: Procs. of 16th International Conference Software Engineering, IEEE Computer Society Press, 159-168.

Appendix 1

Overview of Three MetaCASE Environments: QuickSpec, RAMATIC and Customizer

In this appendix[1] we describe the architecture and main functions of the metaCASE environments *QuickSpec* (Meta Systems, 1989), *RAMATIC* (Bergsten et al., 1989) and *Customizer* (Index Technology, 1987a) – a part of the *Excelerator* tool family (Index Technology, 1987b, 1989).

1 QuickSpec

QuickSpec[2] was originally built to be a Windows-based front-end tool to *PSL/PSA*, but it can also support other languages. PSL/PSA™ is a text-based CASE environment that can be used as a modelling tool and as a repository. It supports the analysis and design, and to some extent also the maintenance, of software. It is a part of the 'product family' *Meta Systems Tool Set*. The other products in this family are a bridging and custom documentation tool *RSI*, a data modelling tool *VIS*, a *Reverse Engineering* tool-kit, a PC-based graphics and analysis tool *Structured Architect (SA-RT)*, and a *Structured Architect* Integrator for multiple SA-RT tools.

New languages can be defined using the MDM tool – a real CASE shell component. This is a facility that can be used to define method specifications for QuickSpec and SA-RT. MDM has never been released as a product but is intended for in-house method modelling. Meta Systems is now a part of the LBMS corporation. LBMS also designed an OS/2 based repository tool, LBMS's *Information Manager* (LBMS, 1991), which was based on the functional structure of QuickSpec but never released.

1.1 The Architecture of QuickSpec

QuickSpec is derived from PSL/PSA; it has two parts: *PSL* (Problem Statement Language) and *PSA* (Problem Statement Analyzer). PSL is a formal language for modelling information systems. PSA maintains the specification database, provides a query system, and produces reports and other documentation. The IS specifications are kept in the database. The meta-database of PSL/PSA contains the rules by which the IS specifications can be checked on entry. PSA

---

[1] This appendix is reprinted from Information and Management 25(1993), Marttiin, P., Rossi, M., Tahvanainen, V.-P. & Lyytinen, K. A comparative review of CASE Shells: a preliminary framework and research outcomes, pp. 11-31, 1998 with kind permission from Elsevier Science - NL, Sara Burgerhartstraat 25, 1055 KV Amsterdam, The Netherlands.

[2] We examined QuickSpec version 1.0 in MS-Windows 2.0 environment.

has four functional components: the command language interface (*CLI*), the analyser for checking the integrity of modifications, the database management system, and a set of report models. Other tools are the interactive query system *QS*, the documentation tool *QuickDoc*, and the report specification processor *RSI*.

QuickSpec originally had a *MS-Windows* based user interface for creating textual statements in the PSL language. It is not a drawing tool like RAMATIC and Excelerator. Its purpose is to make the formulation of correct PSL statements easy by letting the statements be selected from dialogue lists. However, it can be extended to accept statements in any language through its MDM component; it is the CASE shell component of the PSL/PSA family. MDM is a tool for creating method specifications.

The core of QuickSpec is the *Metabase*. It consists of a library of database access routines and a database management system. The library is divided into two sub-collections, which interact with the target and meta-databases. The models of target level and metalevel are based on the data-model called OPRR. MDM is used for generating the meta-database of QuickSpec and other *Metabase*-based tools.

The architecture and connections between these tools are shown in Figure 1.

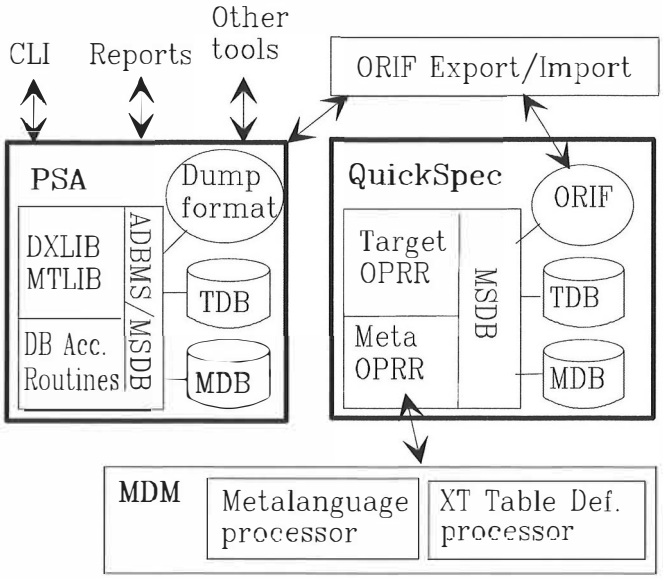

FIGURE 1 The Architecture of QuickSpec, PSL/PSA and MDM

1.2 The Functions of QuickSpec and MDM

QuickSpec has four types of functions: database, encyclopedia, import/export and setting preferences.

Database creation and maintenance tasks are performed through the *Database* menu. It offers the usual commands for creating, opening, closing, and saving databases.

The *Encyclopedia* functions are used for entry and editing of IS descriptions. Descriptions are created using the types included in the defined metamodel. For example, PSL 6.0 contains 18 different object types such as ATTRIBUTE, CONDITION and ELEMENT, a large set of relationship types such as CONSISTS and FLOWS between the object types, and property types such as DESCRIPTION and DOCUMENTATION. Changing the metamodel is done in *Preferences*. Entering new instances of types is accomplished by selecting types and naming them: first, the user chooses an object type and names an instance. After this the legal relationship types for the created object instance can be chosen, and so on.

The *Import/Export* menu is used for transferring information across environments: between QuickSpec databases, and to and from the PSL/PSA system. The user can transfer the entire QuickSpec target database, the last changes was made, or selected statements. The facility uses a special transfer language called ORIF.

MDM has two components: a *Meta language processor* and a *XT table definition processor*. The internal concepts and the structure of a metamodel are made using the *Meta language processor*. The *XT table definition language* describes the external representations for object, property, and relationship types. The *XT table definition processor* connects an external representation to the meta-database.

2 RAMATIC

RAMATIC is a CASE shell developed by SISU (Swedish Institute for Systems Development). It is also a CASE environment, because RAMATIC itself uses the method specifications it creates. The CASE environment is implemented using several description languages. RAMATIC has been in operational use since the end of 1987 in several Swedish organisations.

2.1 The Architecture of RAMATIC

The architecture and functions of RAMATIC are shown in Figure 2. The core is the design object database (*DODB*), which consists of the conceptual database (*CDB*) and the spatial database (*SDB*). The *CDB* stores information about the developed methods; for example, their objects, the relationships between the objects, and their attributes. The *SDB* contains information of how and where on the screen the conceptual objects are graphically represented. A graphical representation is not necessary for all objects.

RAMATIC runs natively on SUN workstations under UNIX, but it can also run in some other environments. The database management of its OS/2

version (which we tested) is based on OS/2 Extended Edition DataBase Manager. The user interface runs under Presentation Manager.

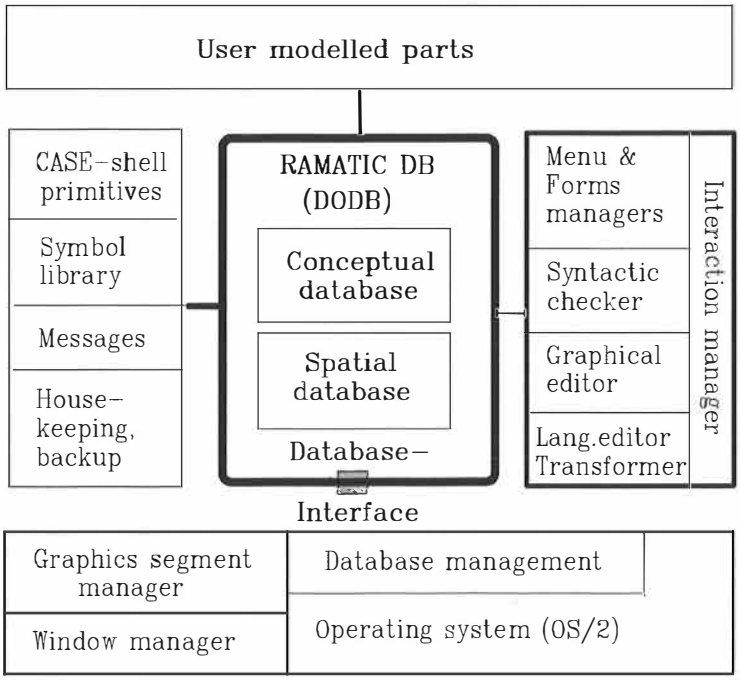

FIGURE 2 The architecture of RAMATIC (Bergsten et al., 1989)

2.2 The Functions of RAMATIC

RAMATIC aims to generalise the shell functions so that they can be used to support different methods. These *CASE shell primitives* are fixed functions that are instrumental in modelling variable methods, for example graphical functions (e.g. draw a line, draw an open curve), window management (create a screen), graphical groups, CDB management, symbol management, syntactical checking and picture management. The connections between menu items and the CASE shell primitives are defined using a *menu description language*.

Interaction management can be defined as management of the parts of the RAMATIC interface when it is used as a CASE tool. It contains a *graphical editor*, a *menu manager*, a *syntax checker*, a *forms manager*, an *analyzer* and a *language transformer*. The *graphical editor* is a window in which models are drawn. Every method has its own menus and forms connected to the *graphical editor* using *managers*. The *syntactic checker* checks the rules of a method.

RAMATIC has a large library of symbol types used in description languages. The methodology engineer can create symbols for a new method using these types or define one using the *symbol description language*.

3 Excelerator and Customizer

Customizer[3] is a product for customising Index Technology's workbench products: Excelerator and Excelerator/RTS. Excelerator[4] is a tool for system definition and analysis, and it also offers possibilities for prototyping. Excelerator/RTS contains some additional languages and associated analysis facilities.

3.1 The Architecture of Excelerator and Customizer

Excelerator partially supports both the SA and IE methodologies. Despite this, the user can define a local procedure when using the supported methods. Accordingly, Excelerator contains description languages (Data Flow Diagrams, Structure Charts and Data Models), and varied representations of these (Gane & Sarson and Yourdon DFDs) found in the methodologies.

The description repository in Excelerator is *XLDictionary*, which can be divided into several project databases. The description languages and forms of Excelerator can be modified by Customizer, which has a repository called *System Dictionary* containing descriptions of *XLDictionary* elements. Customizer consists of four facilities (Figure 3): *Shape Editor* for creating new symbols, *System Dictionary* for maintaining the repository, *Screen Design* for creating forms, and *SLD Interface* for sharing data among different *System Dictionaries*. The *Forms Library* contains forms used in the customised product.

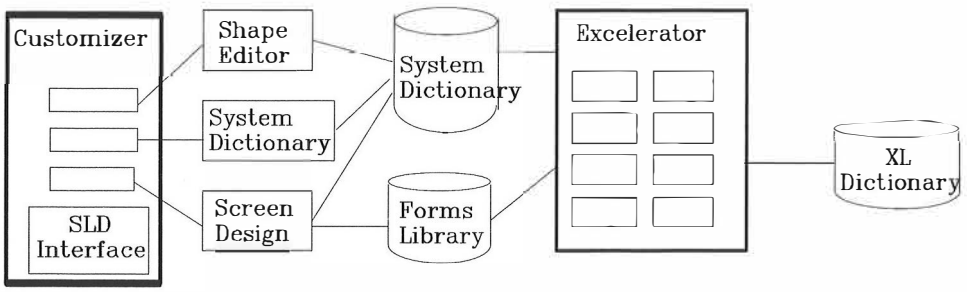

FIGURE 3 The architecture of Customizer (Index Technology, 1987b)

There is also a program called *XL/Programmers Interface* for export/import definition (Index Technology, 1987c). It provides a library of programming functions that enable access to data stored in Excelerator. These functions have to be included in the users' own C programs.

---

[3] We examined Customizer version 1.8.

[4] We examined Index Technology's Excelerator versions 1.8 and 1.9. Intersolv has released Excelerator II product, which contains itself the customization capabilities. One should note that some customization capabilities are improved in this product.

3.2 The Functions of Excelerator and Customizer

Excelerator is a graphical tool. It has functions for drawing models associated with description languages (*Graphics*) controlled by the *XLDictionary*, which supports some groupwork in a multi-user environment. It has facilities for defining and changing 'audit trail' information of projects and users (*Housekeeping*), and for prototyping (*Screens&Reports*). Prototyping means that the user can build form and report models of system descriptions. The analysis functions (*Analysis*) verify the correctness of graphs, dictionaries, prototypes, and connections between different levels. Data produced in the *Analysis* phase can also be gathered and reported (*Documentation*). Also, text processing and project management programs can be used via the *Documentation* tool.

New symbols can be created by using the *Shape Editor*, with drawing primitives (such as 'rectangle', 'circle', 'line', and existing shapes). A symbol is connected to an object type in the *System Dictionary*, and to a form in the *Screen Design*. A *System Dictionary* provides direct access to each SD type's attributes (see Figure 3). The user can examine, add, and modify these attributes or use the information as a basis for reports. *Screen Design* is used for creating the data entry screens in the customised product, based on the different kinds of templates available. For example, the user can choose to use explosion paths for the decomposition. The *SLD Interface* also allows the user to copy system-level data from a customised product to other tools. It thus helps in transferring defined forms, metamodels, and SD types.

References

Bergsten, P., Bubenko, J., Dahl, R., Gustafsson, M. & Johansson, L.-Å. 1989. RAMATIC - a CASE shell for implementation of specific CASE tools, TEMPORA T6.1, SISU, Stockholm.

Index Technology Corporation 1987a. Customizer Reference Guide. Cambridge: Index Technology Corporation.

Index Technology Corporation 1987b. 1989. Excelerator Reference Guide. Cambridge: Index Technology Corporation.

Index Technology Corporation 1987c. Programmer Interface. Cambridge: Index Technology Corporation.

LBMS 1991. Information Manager User's Guide. Pre-Released Version 1.01.

Meta Systems Ltd. 1989. QuickSpec - User's Guide (version 1.0.). Ann Arbor Michigan: Meta Systems Ltd.

Appendix 2

Meta-metamodels of QuickSpec, RAMATIC and Customizer

In this appendix[1] we describe the conceptual basis (meta-metamodels) and the method creation process of QuickSpec, RAMATIC and Customizer, each of which are described in Appendix 1.

1 QuickSpec

1.1 The Meta-metamodel

QuickSpec is based on the *OPRR* data model (Welke and Forte, 1989; Smolander, 1992). OPRR consists of four concepts:

- *object*: A thing that exists on its own and is represented by its properties.
- *property*: A description or qualifying characteristic associated with an object, a relationship, or a role.
- *relationship*: An association between two or more objects. This can not exist if the associated object instances disappear.
- *role*: The link between an object and a relationship: it clarifies how an object participates in a relationship.

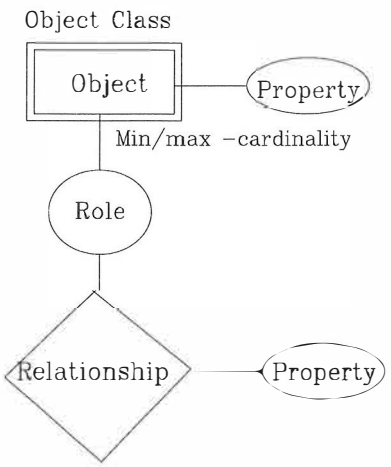

FIGURE 1 The meta-metamodel of QuickSpec

[1] This appendix is reprinted from Information and Management 25(1993), Marttiin, P., Rossi, M., Tahvanainen, V.-P. & Lyytinen, K. A comparative review of CASE Shells: a preliminary framework and research outcomes, pp. 11-31, 1998 with kind permission from Elsevier Science - NL, Sara Burgerhartstraat 25, 1055 KV Amsterdam, The Netherlands.

The *Metabase* is constructed using a limited version of OPRR (shown in Figure 1), in which roles cannot have properties and no more than four object types can participate in a relationship type.

1.2 Using MDM/QuickSpec to Derive Method Specifications

The modelling process can be divided into two main tasks: to define a method in MDM (see Appendix 1: Section 1) and to implement it in the QuickSpec environment. The user has two different ways to cope with the first task.

First, we can follow the longer way. This is useful in most situations, especially when the user wants QuickSpec to use only the types of the new method or the methodology specification. The steps are:

(1) identify and formalise the method specification (for example using the OPRR concepts)
(2) express it in the MDM language
(3) compile the MDM language definition to create a meta-database,
(4) prepare an XT table specification, and
(5) compile it in to the created database

The second way is to use a meta-data base that is created to contain PSL types. This is useful, when the new method can be modelled using PSL. Only steps 4 and 5 need to be done.

2 RAMATIC

2.1 The Meta-metamodel

RAMATIC meta-metamodel (*DSDB*, consisting of the *CDB* and *SDB*) is based on a binary-relationship model. It offers several concepts for sets and properties. It is shown in Figure 2. The object types in *CDB* are described by two base types: FREE-OBJECT and CONN-OBJECT (a relationship between two parts [to and from]). Every object type in *CDB* has a link to one or more object type(s) in *SDB* (spatial). Spatial object types have a link to their picture, to the symbol types in a symbol dictionary, and to groups. Object types can be grouped to a SET TYPE (set) or differentiated to a SUB TYPE (conceptual_sub).

FREE-OBJECTS exist on their own. They usually have a graphical presentation, but this is not necessary. Both CONN-OBJECTS and FREE-OBJECTS can have associations (attributes) stored with them. This binary-relationship core has been augmented. SET TYPES are used for grouping objects. A group can be modelled in a decomposition picture using an ENLARGE-COMMAND. That describes what relationships and objects from the original picture are transported to the decomposed picture.

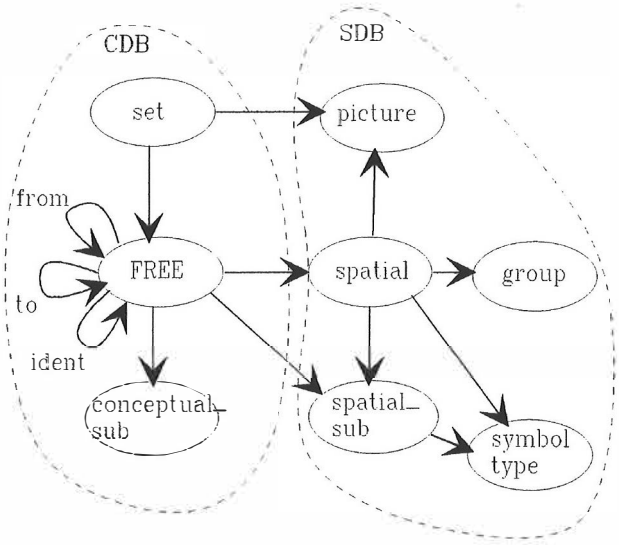

FIGURE 2 The meta-metamodel of RAMATIC

2.2 Using RAMATIC to Derive Method Specifications

The new method is specified in RAMATIC by using *Model, Symbol, Menu* and *Form description languages*. The procedure advances as follows:

(1) create the conceptual metamodel,
(2) produce symbol definitions and link them to the concepts,
(3) produce menus and attach the concepts and symbols,
(4) produce forms for concepts, and
(5) define help texts for the method.

The *Model description language* contains the basis for other definition languages; thus its definition forms the first stage. The model consists of different object types (FREE), connections between them (CONN) and groups of them. The model description contains constraints and semantic checks and the connections between the model types.

When the model definition is ready the model engineer picks up the objects of the model and attaches available symbol definitions or makes new ones using the *Symbol description language*. The symbols are defined by their shape, scale and labels.

When all objects have been attached to their respective symbols, the model engineer makes the menu definitions by creating the appropriate menus, which include items for drawing all object types and grouping and describing them. All menu items also have a shell function attached that is executed when the item is selected. The menus are defined using RAMATIC's *Menu description language*.

For complex information retrieval (e.g. matrices, tables) forms for a method may be defined using the *Forms definition language*. They can be used for cross-referencing design objects and thus for validation.

3 Customizer

3.1 The Meta-metamodel

Customizer's *System Dictionary* is founded on the EAR model. The types used in the meta-metamodel (SD types) are: CONNECTION, ENTITY TYPE, FACILITY, GRAPH CONVENTION, MENU, SCREEN DESIGN, SHAPE, and SYSTEM DEFINITION (Figure 3). Five relationship types can be built between the SD types: CONTAINS, GOES TO, IS DEFINED BY, RUNS and USES. For example, GRAPH CONVENTION contains information of ENTITY TYPES, SHAPES and CONNECTIONS of the current method specification.

In Customizer, a new metamodel is defined as a graph type. The graph type is composed of two main components: GRAPH CONVENTION and ENTITY TYPE. GRAPH CONVENTION specifies which MENUS, rules, and defaults are activated, when the user is creating or modifying this type of graph. An ENTITY TYPE determines how the tool will store and access individual instances.

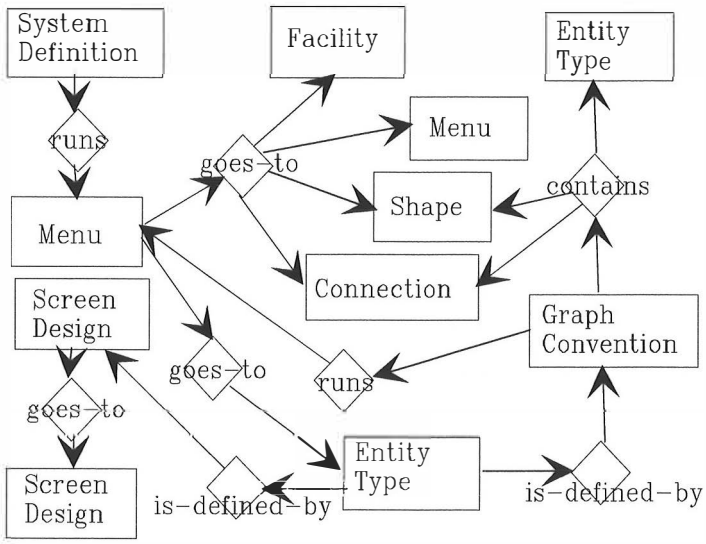

FIGURE 3 The meta-metamodel of Customizer (Index Technology, 1987)

3.2 Using Customizer to Derive Method Specifications

The next seven steps need to be followed in defining a new method for Excelerator:

(1) define symbols for the object types,
(2) define the data entry forms for object and relationship types,
(3) describe the ENTITY TYPES for a graph type and its object types,
(4) describe the CONNECTIONS between object types,
(5) describe the MENUS for ENTITY TYPES and CONNECTIONS,
(6) produce a GRAPH CONVENTION, and
(7) integrate the graph type to Excelerator's *Graphics* and *XLDictionary*.

GROUP CONVENTION contains ENTITY TYPES, SHAPES and CONNECTIONS. The metamodel itself and all its object types are defined as ENTITY TYPES. For example, SMARTIE itself and its object types (Processor, Bank and Channel) are ENTITY TYPES. Relationship types between these object types such as LOOKS UP, GENERATES, RECEIVES and UPDATES are defined as CONNECTIONS. PROCESSOR, BANK and CHANNEL have their own SHAPES and the four CONNECTIONS are represented by lines.

The symbols are created using the *Shape Editor*, and the forms using the *Screen Design* tool. The next steps operate directly on the *System Dictionary*. A new metamodel needs an ENTITY TYPE for saving and accessing data in Excelerator. When this has been created, ENTITY TYPES for all object types will be created. The fourth step is to make the CONNECTIONS and their representations. The data entry screens have to be associated with ENTITY TYPES and CONNECTIONS. After MENUS have been described, ENTITY TYPES, CONNECTIONS, their respective MENUS, and SHAPES are linked together in a GRAPH CONVENTION. The last step is to integrate the method specification into Excelerator's available set of method specifications.

4 The Comparative Review of Meta-metamodel Concepts

Conceptual primitive	QuickSpec	RAMATIC	Customizer
Object/Entity	Object	FREE-Object	Entity
Complex object	–	SUB-object	–
Aggregate object	–	SUB-●bject	–
Relationship	Relationship (max. 4-ary)	CONN-object (binary)	Connection (no semantics)
Relationships to another technique	Relationship (Same concept space)	SET-object or Enlarge	Attribute (explosion)
Role	Role	–	–
Property/Attribute	Property attached to – Object – Relationship – Role	ASSOC-object attached to – FREE-object – CONN-object – SET object	Attribute attached to – Entity (fixed attribute set) – Connection (fixed attribute set)

References

Index Technology Corporation 1987. Customizer Reference Guide, Cambridge: Index Technology Corporation.

Smolander, K. 1992. OPRR - A model for modelling systems development methods. In K. Lyytinen and V.-P. Tahvanainen (Eds), Next Generation of CASE tools, Studies in Computer and Communication Systems, Amsterdam: IOS press, 224-239.

Welke, R. & Forte, G. 1989. Meta Systems on Meta Models, CASE OUTLOOK No. 4, 35-45.

CHAPTER 2:

MODELLING REQUIREMENTS FOR FUTURE CASE: MODELLING ISSUES AND ARCHITECTURAL CONSIDERATIONS

An early version of this paper was published in: J.I. DeGross, J.D. Becker and J.J. Elam (Eds.) *Proceedings of the 13th International Conference on Information Systems*, IEEE Computer Society Press, pp. 9-20, 1992. This version was published in the *Information Resources Management Journal*, 8, 1, pp. 15-25. It is printed by kind permission of Idea Group Publishing, 1331 E. Chocolade Avenue, Hershey, PA 17033-1117 USA.

https://ideas.repec.org/a/igg/rmj000/v8y1995i1p15-25.html

CHAPTER 3:

TOWARDS FLEXIBLE PROCESS SUPPORT WITH A METACASE ENVIRONMENT

A shorter version of this Chapter "Towards Flexible Process Support with a CASE Shell" was published in: G. Wijers, S. Brinkkemper, and T. Wasserman (Eds.) *Proceedings of the 6th Conference on Advanced Information Systems Engineering (CAiSE'94),* LNCS#811, Springer-Verlag, 1994, pp. 14-27.

https://link.springer.com/chapter/10.1007/3-540-58113-8_158

CHAPTER 4:

CAN PROCESS-CENTRED ENVIRONMENTS PROVIDE THE CUSTOMISED PROCESS SUPPORT NEEDED IN METACASE? A LITERATURE REVIEW

This paper was published in: G. Grosz (Ed.) *Proceedings of the Workshop on Many Facets of Process Engineering*, Gammarth, Tunis, 1997, pp. 165-180.

CHAPTER 5:

PROCESS SUPPORT IN METACASE: IMPLEMENTING THE CONCEPTUAL BASIS FOR ENACTABLE PROCESS MODELS IN METAEDIT+

This paper was published in: J. Ebert, C. Lewerentz (Eds.) *Proceedings on the 8th Software Engineering Environments*, IEEE Computer Society Press, 1997, pp. 110-123.

https://doi.org/10.1109/SEE.1997.591823

CHAPTER 6:

HOW TO SUPPORT CASE ACTIVITIES THROUGH CUSTOMISABLE PROCESS MODELS: EXPERIMENTS OF CPME/METAEDIT+ USING VPL FORMALISM AND ISPW-6 EXAMPLE

This paper is accepted to the 5th International Conference on Software Process (ICSP5: Computer Supported Organizational Work), 15-17 June, 1998, Chicago, Illinois, USA.

YHTEENVETO (FINNISH SUMMARY)

Tietojärjestelmien ja ohjelmistojen suunnitteluprosessia yritetään tehostaa menetelmien ja tietokoneavusteisten suunnitteluvälineiden kuten CASE-välineiden avulla. CASE-välineen sovitettavuus organisaation paikallisten suunnittelukäytäntöjen ja -menetelmien mukaiseksi on ongelmallinen prosessi, jota on vaikeuttanut perinteisten CASE-välineiden kiinteä menetelmätuki. Lisäksi suunnittelumenetelmien jatkuva kehittyminen edellyttää CASE-välineiltä joustavuutta. MetaCASE-välineiden tärkein ominaisuus on tilannekohtaisen menetelmätuen mahdollisuus. Tämän vuoksi ne ovat tarpeeksi joustavia tukemaan eri organisaatioiden suunnitteluprosesseja.

Suunnitteluvälineet ovat kehittymässä yhden käyttäjän työkaluista (taskware) ryhmätyövälineiksi (teamware, groupware). Myös metaCASE-välineet ovat vähitellen saamassa ryhmätyöpiirteitä. Suunnitteluvälineissä tarvittavan ryhmätyötuen luonteesta ja laajuudesta ei kuitenkaan olla vielä täysin yksimielisiä, ja sitä tulee edelleen selvittää empiirisillä tutkimuksilla. Myös suunnitteluvälineitä on edelleen kehitettävä, jotta ryhmätyötuen mielekkyyttä voidaan tutkia kattavasti.

Nykyisissä metaCASE-välineissä voidaan määrittää lähinnä mallinnustekniikat (mallinnuskielen käsitteet ja esitysmuodot) ja jossain määrin tekniikoiden keskinäinen integrointi. Prosessien ja työn koordinaation tukeminen ja muokattavuus on oleellinen osa monen käyttäjän suunnitteluvälinettä. Näissä välineissä jätetään yleensä joko suunnitteluprosessiin ja työn koordinointiin liittyvät kysymykset avoimeksi tai niille tarjotaan kiinteitä ratkaisuja.

Mallinnuskielen sopivuus riippuu useista tilannekohtaisista tekijöistä. Tällaisia huomioitavia tekijöitä ovat muun muassa ne, että kielellä voidaan mallintaa sovellusalueella kiinnostavia ilmiöitä ja että kommunikoinnin ja yhteisymmärryksen saavuttamiseksi kieli on laajasti tunnettu projektissa ja organisaatiossa. Myös suunnitteluprosessin esittämisessä olisi oltava mahdollisuus käyttää tilanteeseen sopivaa prosessimallinnuskieltä. Tämän vuoksi sekä prosessimallinnuskäsitteistö että prosessimallin ja sen käyttäjän välinen vuorovaikutus pitäisi nähdä tilannekohtaisina. Nykyisissä suunnitteluvälineissä kuten myös prosessipohjaisissa ohjelmistotyökaluissa (PCSE-välineet) tähän ongelmaan ei ole riitävästi paneuduttu.

Tässä tutkielmassa rakennetaan metaCASE-ympäristöön soveltuva muokattava prosessituki, jonka tarkoituksena on avustaa ja koordinoida mallinnustehtävien suorittamista. Tutkielmassa lähtökohtana on prosessin automatisoinnin sijasta prosessin ymmärtäminen visuaalisen mallin avulla. Tutkimus on luonteeltaan konstruktiivinen, ja se jakautuu teoriaosaan, nykyisten välineiden arviointiin ja muokattavan prosessituen rakentamiseen.

Tutkielman teoriaosassa esitetään monitasomalli (järjestelmätaso, järjestelmän suunnittelutaso ja suunnittelun määrittelytaso) muokattavalle monen käyttäjän suunnitteluvälineelle. Suunnittelu tapahtuu järjestelmätasolla. Järjestelmän suunnittelutasolla esitetään integroidusti suunnittelun tuki eli suunnitteluteknikat (metatietomalli), suunnitteluprosessi (prosessimalli) ja työtehtävien koordinointi (agenttimalli). Suunnittelun määrittelytasolla

esitetään puolestaan käsitteistö ja mekanismit sille, kuinka suunnittelutason konstruktio voidaan määritellä eri tilanteisiin. Tutkimus keskittyy näistä jatkossa suunnitteluprosessin tukeen ja luo perustan prosessimallien ja prosessikäsitteistön (prosessimetamallien) tilannekohtaiselle muokattavuudelle.

Tutkielmassa arvioidaan sekä metaCASE-ympäristöjä että prosessipohjaisia ohjelmistotyökaluja. Ensiksi on tutkittu kolmen metaCASE-ympäristön kykyä määrittää uusi menetelmä. Teoriaosassa esitetty monitasomalli perustuu näihin kokemuksiin ja se ottaa kantaa metaCASE-välineiden puutteisiin. Toiseksi tutkielmassa arvioidaan nykyisten prosessipohjaisten välineiden kykyä tukea prosessimallien ja prosessikäsitteistön muokattavuutta käyttäjän näkökulmasta. Tutkielmassa esitellään kymmenen prosessipohjaista välinettä ja käydään systemaattisesti läpi näiden prosessituki prosessimallien ja -metamallien muokattavuuden näkökulmasta.

Konstruktiivisen osan perustana on monen käyttäjän metaCASE-väline MetaEdit+, jota tutkielmassa on laajennettu muokattavalla suunnitteluprosessin tuella. Tutkielmassa on rakennettu prosessitukiympäristö nimeltään Customisable Process Modelling Environment (CPME), joka koostuu (1) graafisesta editorista prosessimallien rakentamiseen (process modelling) ja tehtävien suorittamiseen mallien kautta (process enactment), (2) GOPRR-p kielestä ja lomakepohjaisista työkaluista prosessimallinnuskielten määrittämiseen. CPME tukee inkrementaalista prosessimallinnusta: sekä prosessimalleja että prosessimallinnuskieltä voidaan muuttaa vähitellen suunnitteluprosessin kuluessa. CPME:n työkalut tuovat entistä joustavamman ja helpomman tavan muuttaa ja testata prosessimallinnuskieliä. Tutkielmassa CPME:n muokattavuutta on testattu määrittämällä graafinen prosessimallinnuskieli VPL ja mallintamalla ISPW-6 prosessiesimerkki. Teoriaosassa esitettyyn monitasomalliin liittyen tutkielmassa on integroitu metatieto-, prosessi- ja agenttimallit - tosin ainoastaan kaksi ensiksi mainittua on rakennettu käsitteistöltään muokattaviksi.