CHAPTER 1

INTRODUCTION TO THE RESEARCH

1 Introduction

The last thirty years have seen a rapid growth in the use and importance of information systems in society, coupled with a growing discontent of the failures and problems experienced with these systems. Before half-way through this time period, it was already widely recognised that we were facing a 'software crisis' (Bro82), whose effect could be likened to that of an epidemic whose symptoms appear only slowly: the situation was already bad when it was recognised, it was getting worse, and there was no cure in sight. The central observed problem was a massive backlog of projects that had fallen behind schedule, and systems that were of low quality and suitability, with virtually no documentation.

The search for a solution was focused in three directions: **analysis** of the whole process and concepts involved in building information systems (e.g. Che76, Bro82), application of more rigour in the early stages of projects by following explicit **methods** (e.g. Gan79), and the **documentation** of the development of systems, often using computers (an idea going back as far as the early seventies (Bub71)). The branch of methods grew with astonishing rapidity, largely subsuming the other two branches, and producing huge numbers of methods of increasing complexity. Aside from leaving practitioners stranded in a 'methodology jungle' (Avi88), and for a long while forcing academics to limit their research to classifying the methods (Oll82), rather than examining how they performed in practice, the growth of methods easily outstripped that of the computer tools that were built to help implement them.

The very early, text-based Computer-Aided Software Engineering (CASE) tools such as PDL (Cai75), PSL/PSA (Tei77), SEM (Tei80) and SREM (Alf77) had allowed changes to the method supported, which gave users some possibility to maintain tool support in the face of the rapidly changing methods. However, newer methods and tools had adopted graphical representations and interfaces. Whilst these were substantially easier to use, they were more complicated to specify, and thus CASE tools were no longer able to provide the user with facilities for changing the method they supported. The CASE tools, heavily outnumbered by methods, were thus forcing the users to adopt their built-in method, rather than supporting the methods from which the organisation was already starting to see benefits. Conversely, the organisation could continue with its own method, substantially weakened by the lack of computer support, or even build its own CASE tool, a heavy investment in a venture in which it had no experience, and could only make financially feasible by making the result available commercially.

These conflicts, of course, did nothing to improve the image or practical benefit of the expected CASE revolution (You86). CASE had been unrealistically trumpeted to be the 'silver bullet' that would solve all information systems development (ISD) problems. All too often, the response to the failure of CASE to provide such a solution was to blame the method or the tool. This of course led to the development of yet more methods or tools: hardly likely to improve the situation. The software crisis meanwhile continued, and no other cure was

on offer, so all projects of any size and complexity were still either looking to using CASE, taking it into use, or licking their wounds after an unsuccessful experience. Only a few were realising anything like the 'full' benefits, and their situation was by no means the most common: they were:

- strategy-oriented and aware (Sil90),
- had a well-defined method already in use (Par90), and
- this method was supported by an available CASE tool (Aae92b, Kus93).

A similar mixture is recognised by Le Quesne (LeQ88, LeQ90), and whilst McClure (McC89) lists 15 mostly tool-centred problems, he too recognises that organisational factors are as important as these. Not even all the organisations fulfilling these conditions successfully adopted CASE: as many researchers point out, the successful adoption of CASE is "dependent on a complicated mix of organisational and technological conditions" (Aae92a).

Taking these conditions, however, we can see that ensuring the satisfaction of the first two is outside the scope of research into CASE tools, and our aim must therefore be to help those organisations which fulfil these by providing a CASE tool which will support their method. As has been stated, it is a practical impossibility to build a new CASE tool for every method, and there is no suggestion that everyone will begin to use the same method, or even one of a small set of methods (Bub92). Certainly attempts to consolidate a group of methods, as the Unified Modelling Language (Boo97) does for object-oriented methods, are worthwhile. They do not, however, signal the end of method development, but rather the maturing of one cycle. There will certainly be further versions of the UML, and we would expect to see a new paradigm of methods rising up to take the place of object-oriented methods, just as these replaced structured methods (You86). New methods of course require new tools, and it is ironic to note that the UML has caused perhaps the largest ever upsurge in releases of CASE tools.

As we have observed, building a whole CASE tool from scratch is a large and complicated project, significantly too slow to keep pace with method development. The solution to this conundrum thus lies in a CASE tool which can be customised to support any method. Two possible approaches to this are perceivable: a CASE tool could be designed and built modularly, so that the minimum coding effort is required to change the part concerned with a particular method; or a CASE tool could have its method as data, rather than as code, and functionality could be provided for altering this method data, in the same way as was done in the early text-based tools.

The former solution is the one largely adopted by industry, but is flawed from the users' point of view: only the vendor can make the changes, and the cost of such changes is high. Whilst the reduction in work to make a CASE tool for a new method is significant (one manufacturer claims reuse as high as 90% (Rus94)), the rate of such adaptation has still proved insufficient to satisfy users' needs. Furthermore, the cycle from requesting a change to the method support to using the modified tool is painfully long, and the customer is left highly dependent on the vendor.

The latter solution, called CASE shells (Bub88) or metaCASE tools (Ald91), has produced promising research prototypes and a few somewhat limited commercial products. These have not yet been widely taken into use, although the use of Systematica's Virtual Software Factory metaCASE tool (Poc91) by IBM in building its BSDM support tool (Hai92), and again by Heym and Österle in the construction of their MEET method engineering environment (Hey93b), provides practical proof that such tools can be useful. Comparisons of these metaCASE tools (e.g. Mar93, Gol93), have revealed that the process of metamodelling (Tei80, Bri90) — configuring the tools to support a method — could be improved, as could the coverage of the support for the method in the configured tool.

The research in this thesis aims towards metaCASE tools which would better answer the needs and criticisms of CASE tool users, in these two particular ways:

- the process, concepts, and tools for metamodelling should be improved;
- the metaCASE tool should be capable of being more easily and accurately configured and tested for a wider range of interlinked, evolving methods.

Work towards these goals will improve the existing success of metaCASE in remedying observed CASE problems, which in turn will improve the whole ISD process, bringing corresponding benefits in the information systems which are playing an increasing part in our society. In particular, the first goal will speed and ease the process of improving the methods we use to develop ISs, and the second goal will allow better implementation and support of these methods in CASE tools.

In the next section we will look at the background and terminology of this field, thus briefly presenting the conceptual framework within which the research takes place. In Section 3 we examine the current situation of the field, its problems and some of the proposed solutions. These motivate and provide our research problem, discussed in Section 4. The methodology with which the research is carried out is justified and presented in Section 5, along with its application in this thesis. A short summary of each paper is presented in Section 6, followed by a brief overall conclusion and directions for future research.

2 Background

2.1 Terminology

The terminology in this field is in a sorry state, with homonyms and synonyms causing confusion and endangering communication among researchers, and between research and practice. It is to be hoped that the work of groups such as FRISCO (Lin90) will help produce de facto standards for the terms to be used and their meanings: in the mean time, we are forced to explicitly define the

terms used in this thesis and their meanings, before we go any further. Only the barest justification is given for most of these definitions, although it is hoped that they form a clear and mutually consistent and supportive set. Further information about the later topics is to be found in the rest of this introduction, especially Section 2.4.

ISD: Information systems development

Judging from the acronym, one would be justified in assuming that ISD represented only the development phase of systems engineering, and need not necessarily be systematic. The term ISD has, however, accumulated a rather more advanced meaning. We will follow this trend, and define ISD after the fashion of Welke (Wel83) and Lyytinen (Lyy87b):

*Information systems development is a change process taken with respect to an **object system** in an environment by a **development group** using **tools** and an organised collection of **methods** to produce a **target system**.*

The use of 'system' in object and target systems is to be taken in the most general sense: we may be writing code, designing a database, modelling a method etc.

Software / systems engineering

Nunamaker et al. (Nun91) list several definitions of software engineering. For our purposes the definition of (IEE83), short and to the point, will suffice:

Software engineering is the systematic approach to the development, operation, maintenance, and retirement of software.

Systems engineering is a superset of software engineering that also includes, inter alia, engineering of businesses' information structures.

(ISD) method

A method is a way of carrying out ISD. Whilst also applicable to unformalised, subconsciously applied ways of working, the term is more generally used to describe a formulated process and set of intermediate representations and rules for progressing in the course of ISD. The term 'methodology' has caused some confusion, with one school of thought following etymology and Maynard (May39) to have it mean 'the study of methods' (e.g. Har94), and another school using it for 'a body of interlinked methods' (e.g. Kum92, Oll82, Oll86). Similarly, there are differing opinions as to what constitutes a method: one kind of diagram and instructions for how to build it? a set of interlinked diagrams? any

procedure, possibly using many sets of diagrams? Various other classifications and names have been proposed, including 'fragment' and 'technique' (e.g. Har93, Har94, Bri90, Slo93), but none of these has yet been widely accepted. In this thesis, then, we will use 'method' to include both those concerning single and multiple diagram types, specifying more exactly when we are talking only about a single diagram type or multiple linked diagram types.

Another source for information about what constitutes a method is the books which present and describe individual methods. Such a book often includes three types of information: worldview (a way of thinking (Wij91)), data model (a way of modelling), and process model (a way of working). In current CASE tools really only the data model has been implemented: the rest are largely left up to the user. For this reason the term 'method' in connection with CASE tools, at least in normal usage, largely refers to the data model part of the method; changing the other two parts is not seen as changing the method supported by the CASE tool. The data model part then is also the central part of a method from the viewpoint of this thesis, concerned as it is with CASE tool support for methods.

An example of a simple single diagram type method is Data Flow Diagrams, which together with other linked diagram types forms part of the wider SSADM method (Gan79). More recently, there have appeared many object-oriented methods (e.g. Boo91, Rum91), which often include diagram types borrowed from the older structured methods.

Method engineering

As method engineering is only a specialised kind of ISD (see e.g. Lyytinen's (Lyy87a) definition), it can usefully be defined by extension of the definition of software engineering (IEE83):

Method Engineering is the systematic approach to the development, operation, maintenance, and retirement of methods.

Until very recently, actual method engineering could hardly have been said to exist as a discipline, with published research covering only concepts and principles (e.g. Bri90, Kum92, Hey93b), the role of the method engineer (Bub88), or studies of how metamodelling takes place in practice (Tag90). Methods were certainly developed, but the process of their development was not well defined, and largely haphazard. This has been one factor leading to the development of the current ill-structured, over-populated 'method jungle' (Avi88). Lately, empowered by a greater theoretical and practical understanding of methods and their development, research has started to be published on the process of method engineering (e.g. Tol93, Tol95). The term 'method engineering' is unfortunately often used when people actually mean 'metamodelling'.

Metamodelling

Metamodelling in general is the modelling of the languages we use to model with (cf. Bri90). In ISD it is used for the process of making a model of an ISD method: metamodelling is thus just method modelling. In many ways, the process of modelling a method is very similar to that of modelling a system, as has often been recognised (e.g. Har94, Smo91b). This similarity has motivated reuse of concepts, methods and tools between the areas, e.g. OPRR (Wel92, Smo91b), ER (Che76), and NIAM (Nij89) have all been used as data models for both methods and information systems, and the MetaEdit environment (Smo91a), which uses OPRR, uses the same graphical tool to manipulate both models of systems and models of methods.

Whilst most metamodelling is for the purpose of implementing a method in a *metaCASE tool* (see below), methods are also modelled to examine their relationships to each other (e.g. Oei94, Ros94b) or how well a set of methods would work together (Kin94), or in a given tool (Bri89).

CASE (tool)

Originally standing for Computer-Aided Software Engineering, more recent recognition that the theories and tools of CASE can be applied to a wider area of ISD, notably to business systems, has led to the replacement of 'Software' by 'Systems' in the acronym. Unless otherwise specified, a CASE tool is generally thought of as supporting one fixed method.

CASE can be divided depending on the phase of the ISD process, and one common way is to use 'upper CASE' to refer to the upstream parts of the development such as requirements analysis and design, and 'lower CASE' for the later phases. Integrated CASE or I-CASE includes a well-linked combination of both upper and lower, and also generally aims further at support for the extreme ends of the development process: very early project definition, and programming and maintenance of the resulting IS. Whilst some tools still claim to support I-CASE, most manufacturers have taken a step back from this area, realising they cannot deliver what they promise (Lou92 p.379).

Although CASE as a term is by definition rather broad, including any computer support for systems engineering, in usage the term has largely come to mean fixed single method upper CASE tools. We shall follow this convention in this thesis, without at all underestimating the value of computer tools in other parts of systems engineering, or the need for integration of CASE tools with such other tools.

Examples of CASE tools abound: even back in 1988 they numbered in the hundreds (Bub88). Tahvanainen and Smolander (Tah90) provide a good selection of references on CASE and CASE tools. Some well-known examples are Software Through Pictures (Was86), Excelerator (Exc87), Rational Rose, and Select OMT.

MetaCASE (tool)

A metaCASE tool or CASE shell is a CASE tool whose method support can be changed (the former name (e.g. Ald91) appears to be replacing the latter (e.g. Bub88), being more accurate and descriptive). A true metaCASE tool allows the method to be completely changed for a new one, but many CASE tools are now exhibiting some metaCASE functionality by allowing additions and minor cosmetic changes to their existing method support. A metaCASE tool need not include within itself the functionality for defining the new method: that may be defined elsewhere and compiled externally into a form that the metaCASE tool can accept. In such cases we generally talk about a 'generic CASE component' and a metamodelling component.

In this thesis, we do not accept as true metaCASE anything which requires programming of resulting CASE tool operations in a programming language or something close. This follows from the term metaCASE itself: as a metalanguage is still a language, even if one level higher, metaCASE must still be CASE: programming is not generally recognised as CASE, therefore systems requiring programming should not be regarded as metaCASE. Whilst we do recognise that such systems too have their benefits, it becomes impossible to decide how much programming should be allowed (or else we risk classifying C++ as a metaCASE tool!), and there are enough metaCASE tools which involve no programming to show that programming can be eliminated completely. This rules out systems such as PPP (Gul92) and GOODSTEP (GOO95), which are better regarded as libraries or frameworks to support CASE and help add new method support.

We do however allow metamodels to be written in textual languages, as these do not contain the control structures found in programming languages. Similarly, we would allow rule and constraint expression in logic languages or similar, and the possibility of programming to *extend* CASE tool behaviour, providing that basic standard CASE functionality can be achieved without it.

Examples of metaCASE tools are Excelerator's Customizer (Exc87), RAMATIC (Ber89), MetaView (Sor88), VSF (Poc91), IPSYS Tool Builder (Ald91), and Paradigm Plus CDK.

MetaCASE environment

The term 'metaCASE environment' (Kel94b) has been used for a system which supports metamodelling (modelling of methods) in the same environment as modelling, and itself produces the metamodel and inputs it to the metaCASE tool. Examples of metaCASE environments include MetaEdit (Smo91a), KOGGE (Ebe97), and MethodMaker, although all of these require some degree of user intervention in the transformation of the modelled method into a functioning metamodel.

CAME (tool)

CAME is computer-aided method engineering. CAME thus does not necessarily involve any metamodelling, or any metaCASE. Early 'CAME' tools supported contingency-based selection between several methods (e.g. HECTOR's SESAM (Sav90)). More recently it has been possible to record the various parts or fragments of methods and build new methods from them (Hid93, Har94) (note this involves some degree of metamodelling). These tools, if they can truly be called CAME tools, thus support only a very narrow area of methodology engineering.

CAME environment

A true CAME environment, if the generated methods are to have computer support, would have to include an integrated metaCASE environment, in order to check the engineered method in use, and allow incremental method engineering (Tol93, Tol95, Kel94a, Ros94b). This would remove the need for user intervention mentioned for the existing metaCASE environments. Currently none exist by our definitions, although Decamerone (Har97) is a close research prototype.

Thus the more CAME functionality a metaCASE environment has, the more it qualifies as a CAME environment. Such CAME functions are described in e.g. (Mar96), and include support for querying and reusing types in metamodels, updating models when metamodels change, and automatic checks and guidance for the metamodeller. Similarly for a CAME environment to qualify as 'multi-user', it should allow multiple simultaneous modellers and metamodellers.

2.2 Definition of the ISD process

There are many models of the ISD process that largely agree on the basic definition of ISD which we are using and thus could be used here (e.g. Aur88, Smo91a, Mar92). These models, however, tend to be complex and unclear in terms of outward appearance, and, possibly related to this, some blur over some important conceptual differences in their attempts to be universally applicable.

A major factor leading to the intimidatingly complex appearance and lack of clarity of all of these descriptions is their use of only binary relationships. Information Systems Development is an inherently complex process, involving multifaceted interactions between several parties, human, technological and abstract. At each stage it is possible to identify around five parties, and it would be possible to show a meaningful relationship between any two of these: a little arithmetic informs us that we could thus draw 20 relationships — and that assumes that from any one party to another there is a maximum of one relationship, whereas many authors have several.

Whilst each of these relationships could be a fair representation of an actual relationship or interaction, a multi-part or n-ary relationship probably better serves our purposes, expressing the basic idea that there is one interaction or process underway, that involves many parties each acting in a different role. Thus, while not abstracting away any more detail than do the other models, we can produce a diagram that is substantially simpler to understand, and hopefully reveals a truer picture of the underlying process.

The Information Systems Development process is generally at each stage a move from a **less precisely defined model** (object system) or concept of a system to a **more precisely defined model** (target system). This move is carried out by a **modeller** or development group, in and possibly aided by a certain **environment** and its tools, and the process and resultant model follow the rules of a given method language or **metamodel**. The actual move itself — the relationship — can be understood as an instantiation of the *way of working* from Wijers' set of terms (Wij91): *how* the modeller performs this move.

Taking the terms in bold as the central concepts or entities in play, we can thus represent this as a 5-part relationship in the form of a cross: for convenience the environment, being the central location of all activity and the cross-roads through which almost all information passes, is represented directly over the centre point of the relationship. It is important to understand that the model contains only *one* relationship, which involves five objects in different roles. This relationship and its roles forms a *pattern*, which can be applied to the several different kinds of modelling that occur at different levels and stages of ISD.

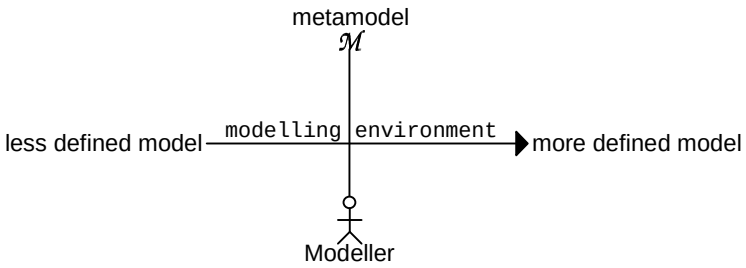


FIGURE 1 Modelling in ISD as a 5-part relationship

Of course, any one of the parties may be complex: we may have many modellers working together, for instance. Similarly, the parties are to be understood as situated in their own environment: the modeller within an organisation, the modelling environment in an operating system with other tools, and particularly the two models within their own environments. Thus for instance a modeller draws on not only a specific requirements definition but also his own and others' knowledge of the target organisation: the terms in the diagram denote the *central* concepts of the parties involved, and are not intended to form exhaustive descriptions. Thus the application of 'less defined model' and 'more defined model' to a situation may result in something that is not normally called a model: they may be metamodels, mental ideas, program code etc. The 'more defined model' may also include things not found at all in the less defined model, e.g. an important part of early phases of ISD is the

identification of extra aspects of the problem that should be considered. Basically, though, the move from left to right always represents some kind of process of increasing formality, precision, definition or information.

Sometimes the assignment of 'less defined model' and 'metamodel' can be difficult to determine: the simple rule is that the 'metamodel' defines what kind of resulting models are *legal*, in general, whereas the 'less defined model' is our measuring stick for judging the *accuracy* of the resulting model. Similarly, in many situations the metamodel is a fixed part of the modelling environment, often inseparable from it. In this thesis, concerned as it is with modelling environments whose metamodels can change, it is worthwhile making the distinction.

The second problem with many of these earlier frameworks is the laxity with which the meaning of the relationships is handled. Many frameworks, for instance, contain three levels, 'Metamodel', 'model', and 'IS', showing them as being three points along the same dimension. The situation, however, is not that simple: the model and IS are both different representations of the same thing, whereas the metamodel is the language in which the model is written. Conversely, we could start with acknowledging that metamodel is the language for the model, but note that the language for the IS is some programming language, rather than the model. Similarly, other frameworks show 'metametamodel', 'metamodel' and 'model' correctly as being three points along the same dimension, but incorrectly use the term 'metametamodel' for something other than the language in which metamodels are written. This kind of laxity in relation to the misuse of the 'meta-' prefix is also noticed by Leppänen (Lep94).

A more accurate model of the interrelationship of metamodel, model and IS, and their production, would look like Figure 2. Note that the link between first and second relationships is formed differently than between second and third: the shared object plays different roles in the second relationship of each pair. This makes explicit the difference mentioned above which was missed in many earlier frameworks.

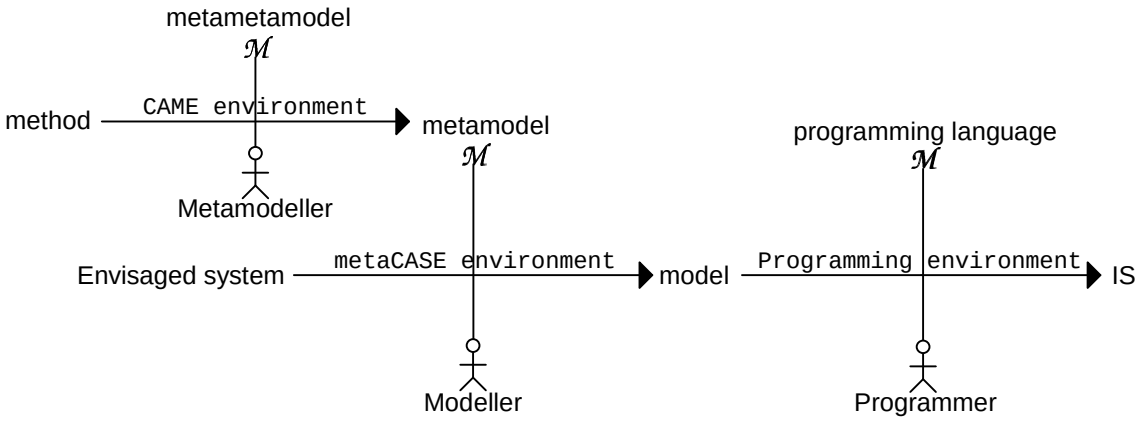


FIGURE 2 Three levels of ISD showing inter-level relationships

We could add further 'cross' relationships to show the development of the method or metametamodel, but these processes tend to be more abstract, less

well defined, and as yet poorly understood. Similarly, we could add complexity by showing that many of the parties evolve over time, but that is not our main focus of concern here. We could also draw another cross relationship showing the IS in use, with the resulting IS system functioning as the environment, but our model is probably not particularly well-suited to that: what would be the less defined and more defined models there, and is there anything we can consider as being a metamodel or language, other than some non-discrete part of the IS itself?

This 5-part relationship model, then, is particularly well-suited to the stages of modelling methods, and of modelling systems — processes whose similarities have been recognised before (Bri90), and on which we concentrate in this thesis.

2.3 Processes and products

Books which present and describe individual methods often includes three types of information:

- worldview: information about the overall philosophy and background of the method (way of thinking (Wij91));
- data model: information about the individual concepts of the method including their representations (product metamodel, way of modelling);
- process model: information about the order of steps in applying the method (way of working).

Of these, CASE tools only really support the second: the worldview is left up to the user, as in most cases is the process model. Attempts to support the process model have largely been confined to rules for low-level operations, which many would class as part of the data model. Process centred environments in contrast focus on the process, either at a low level, in which case they also include some CASE functionality and a data model, or at a high level, in which case they generally do not include CASE functionality, but rather could call an external CASE tool. The low-level process support has in practice often been found to be too restrictive: Verhoef and ter Hofstedte (Hof96) noted that process support for ordering tasks like 'creation of an entity' is not feasible, arguing that it is inappropriate to restrict the freedom of specialists in their work.

It has often been noted that modelling processes is itself a modelling of a system, and indeed many mainstream CASE methods include diagram types for modelling processes, e.g. State Transition Diagrams. Thus we have process models, languages in which they are expressed (e.g. State Transition Diagrams, Petri nets), and a higher language in which we define such languages. This corresponds to the situation with product (data) models, i.e. model, metamodel and metametamodel. The difference appears when we consider the people and phases involved: a process model is made based on a method, probably by the same person who models the data model of that method. The resulting process model is then used in parallel with the meta (data) model, to guide the modeller in developing models of the envisaged system.

There are thus three stages concerning processes in a metaCASE and CAME environment, compared with two concerning products (see Figure 2: note that we omit the last stage there that produces an IS, as this does not take place in the metaCASE environment). The first two stages (metamodelling and modelling) are of the same nature in both, and we can prefix them with 'process' to make the distinction clear. The additional third stage is new, and involves the application of the completed process model. These three stages are shown in Figure 3: the model of ISD described above includes no role for process guidance, thus we add a thick grey arrow to show it.

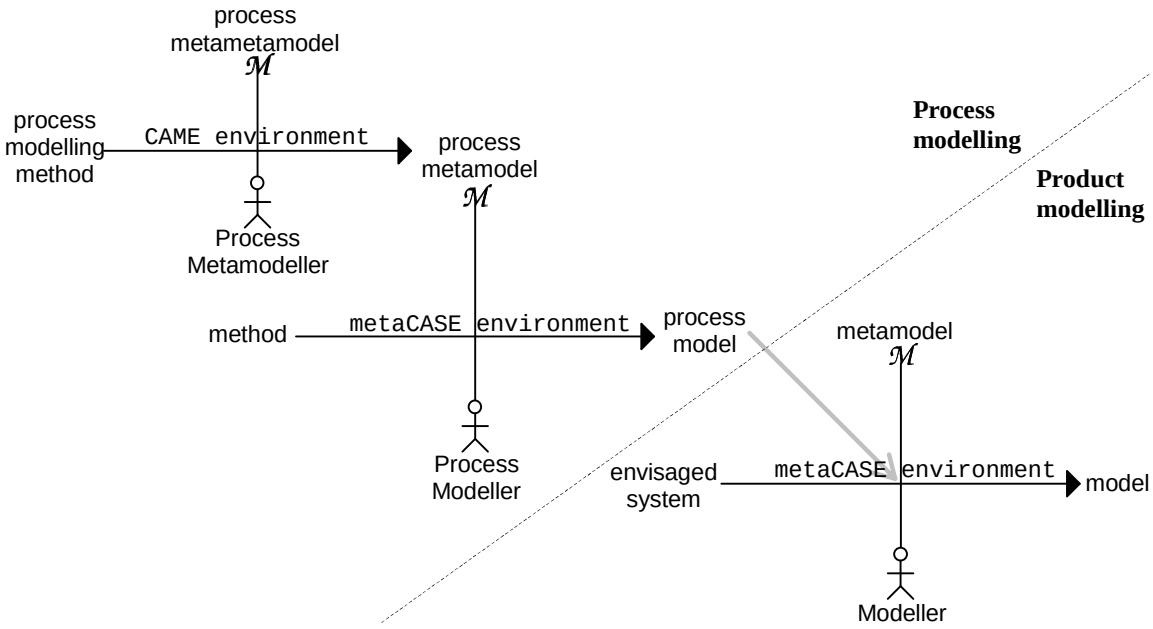


FIGURE 3 Process model creation and application in a metaCASE environment

The third stage here is thus the *second* stage from Figure 2. It should be emphasised that the first two stages here do not take place in parallel with the first two stages in Figure 2. It would be more natural to suppose that the process metamodeller defines one or more process metamodels once, and these are seldom changed. Subsequently for each method to be modelled the (data) metamodeller and process modeller (possibly the same person) work to produce a meta (data) model and process model in parallel.

This is of course only one view: it could be simpler, for example if the process metamodel were fixed in the metaCASE environment. Alternatively it could be more complicated: it might be possible to make a 'method-independent' process model, a series of steps that is good to carry out regardless of which method, or rather meta data model, is being used; examples would be parts of standards such as SEI CMM (Pau93), ISO/IEC 12207 (ISO95) and IEEE 1074 (IEE95). The grey arrow above would thus include some work for the process modeller or project leader, in attaching the concepts of the meta data model to be used into the various steps of the generic process model. In addition the process model could be modified at that stage, for example removing a step that does not apply to this project. Thus the process model

produced at the end of the second phase above would be used as a *template*, which must be integrated with a meta data model and possibly modified before being used. Such an approach has indeed been followed in MetaEdit+ (Kos97), but shall not be discussed in this thesis.

An important part of or addition to process models is an agent model: which people (or other non-human agents, e.g. programs) are involved in which processes. This is often extended by the concept of a user role: each person may play a multitude of roles (e.g. project leader, designer), and for many processes it is more important that someone able to play a given role performs a process, than that a specific person performs it. It is thus more sensible and flexible to specify that 'a designer' will draw an ER diagram, than that 'J. Smith' will draw it. The current understanding of such agent models is however not very advanced compared to that of data models, and improving it falls outside the limits of this thesis.

2.4 Concepts and representations

Whilst we have spoken above largely of the metamodel and model as containing concepts, they also contain *representations* for these concepts. A representation in a model is in general a graphical symbol with labels showing the values of properties of the given conceptual model component. Representational information includes facts such as the exact x and y co-ordinates a particular object has in a diagram, or the width and font used for an object forming a column in a matrix. In a metamodel, the symbol is specified, along with information about which property values are displayed where and how. This distinction between conceptual and representational aspects is important, and can be considered as a dimension of information in CASE (Smo91a), orthogonal to the type-instance dimension already covered. As can be seen, the model presented above works equally well for representational information: the meta-metamodel is then the language or way of representing symbols.

An explicit separation of conceptual information from representational allows us to maintain the same conceptual information, and display it in several different representations, e.g. as both a diagram and a matrix, or as two diagrams with different layouts. For instance, a Class Diagram graph could have two different diagram representations, one stressing the inheritance hierarchy and the other the aggregation hierarchy: the underlying conceptual graph would be the same for both. Most of the conceptual objects would then have representations in both diagrams, whereas the conceptual relationships would mostly have a representation in only one diagram or the other.

3 Current situation and related research

Having set out our terminology and conceptual framework, we now use these to aid us in examining the state of the art in CASE and its sub-disciplines. In particular, we are interested in the problem areas that can be identified, for it is in these directions that our efforts should be concentrated. We look first at the general state of tool support for CASE, metaCASE and CAME, including descriptions of an extensive set of existing commercial and research tools, and then more closely at the results of empirical research on CASE tool use.

3.1 Current tool support

At present, there are hundreds of CASE tools on the market, only a very few metaCASE tools, and no true CAME environments, excluding research prototypes. Most of the CASE tools support the use of a fixed method, almost always graphical, in which representations of the information system can be drawn as part of the analysis, design and documentation of the system. The structure of these diagrams and the information contained in them is recorded in the 'data dictionary' or 'repository'. Textual reports can be produced from the contents of the repository, and the syntactic correctness of the data there can be checked on a fairly simplistic level. Some more advanced CASE tools, especially those intended for the rapidly-growing area of business process engineering methods (Dav90), include support for other non-graphical representational paradigms such as matrices or tables, e.g. Anderson Consulting's Foundation (Fou87a, Fou87b). We will not consider process centred environments here, as the majority are prototypes or do not themselves qualify as CASE tools; Marttiin (Mar97) examines a number of such environments from the viewpoint of integrating process support with metaCASE.

Current metaCASE tools separate the metamodelling part from the CASE part, with the description of the method being written in a textual language, possibly compiled to some other form, and then fed into the configurable CASE tool, which then supports the new method. Some more advanced tools provide partial graphical support for the metamodelling process, e.g. MetaEdit (Smo91a) and Mark V MethodMaker. In tools of this type the graphical metamodel is made with the same CASE functionality as the resulting CASE tool, but using a special 'metamodelling' method (whose metamodel has been bootstrapped from a textual description, and may have special links to CASE tool functionality reserved only for that method). The resulting metamodel is first transformed to a textual language, which is then compiled etc. as above.

This separation of metamodelling and modelling has an important drawback: it is not possible to interactively test the results of metamodelling immediately, because of the long transform-compile-link-run cycle to move from metamodelling to modelling. This is a severe hindrance to the metamodelling process, as if a problem is spotted whilst testing a method, the

user must exit his model, restart the metaCASE tool, read in the metamodel, edit it, transform, compile, and link it to the resulting CASE tool (these steps often require separate commands, manual text editing and external tools). The resulting CASE tool can then be restarted, and the existing models must generally be explicitly updated to use the new metamodel before the change can be tested.

None of these tools provides real support for method engineering: in addition to the difficulty of testing described above, they generally have no support for reuse. Conceptual reuse of method components is difficult or impossible, and tool support for finding reusable components and reusing them is non-existent.

There are however some early CAME tools, although the range of the CAME process these cover is at present severely limited. Most concentrate on selecting a subset of a single method, rather than integrating among several methods, or building new methods. Lack of integration with the metaCASE and resulting CASE tools hampers the incremental nature of method engineering (Tol93, Kel94a, Tol95).

Much constructive research in metaCASE has failed to make full use of results from previous research efforts, both commercial and academic. Any number of projects have implemented a simple single-user metaCASE tool with very basic functionality, despite the large number of existing research prototypes and even commercial tools. Where researchers have referred to existing tools, they have often examined only versions several years old or weaker tools, despite the availability of more recent information and published material. In an attempt to rectify this situation, in the rest of this subsection we will examine a rather large set of metaCASE tools. Even so, exhaustive coverage of the exact current state of all existing tools is not possible: the field is constantly progressing, and commercial tool manufacturers in particular are wary of giving information to researchers, whom many view as potential competitors. These descriptions thus represent at best a snapshot of available information.

We roughly divide the tools into full products (released and used, whether commercial or not) and research prototypes (i.e. never fully released or used). In addition we list some scientifically interesting tools that are currently unavailable, and some tools that have been mentioned elsewhere as metaCASE but which we feel do not qualify. As there currently exist no true metaCASE environments with good CAME support, we will not look at CAME tools here: instead, the issues of CAME functionality are covered in the following section. An exception is made in the case of the Decamerone environment (Har97), which is included below as an example of those CAME tools that do not fulfil our criteria for metaCASE.

The tools are examined in alphabetical order: after each tool name we give the manufacturer or university that developed it: where tools have been commercialised or changed hands, several names are given in chronological order. Because many of these tools are not covered in scientific publications, and trade publications are often several years old, we have included material from the manufacturer's own WWW site with footnotes to the URLs, or from

other promotional material. Where dates are not given, the information is from July 1997.

The descriptions are not intended to cover each tool fully, but rather to help illustrate the various ways in which metaCASE tools work, and how effective these ways are. The tools thus provide concrete examples of the general discussion of metaCASE, and an overview of what related research, both academic and commercial, has achieved. After the tool descriptions, we summarise the most important tools in a table.

Full products

MetaEdit (now MetaEdit Personal) (Jyväskylä/MetaCase Consulting)

MetaEdit (Smo91a) consists of a generic CASE tool part whose method support is provided in binary metamodel files. It uses a textual metamodelling language based on OPRR (Smo91b). It includes graphical tools for generating this language for both conceptual (using graphical OPRR) and representational (using a separate symbol editor) parts of metamodels. These two parts of the metamodel are then joined by hand into one text file, and this is run through the Moffer metamodel compiler to generate the binary metamodel file which drives the generic CASE tool part. If a metamodel is changed, existing models based on it can be explicitly updated to use the new metamodel. This ease of metamodelling is the main advantage of MetaEdit: it is the easiest metamodelling tool described here.

MetaEdit has several limitations, mostly in its CASE functionality. We describe them here in some detail as they form areas for improvement in the development of MetaEdit+ covered in this thesis. Models are limited to a few tens of graphs, all of the same graph type, and there is no linking between models. This prevents proper implementation of most methods, where there are several integrated graph types. An odd implementation of relationship metamodelling requires creating several near-duplicates of relationship types in some not infrequent cases (Kel95), also complicating CASE use with extra types. The conceptual-representational distinction is rather weak: only objects may have more than one representation, and each model file is conceptually one large graph, of which subsets are visible in different diagrams, contrary to the user perception of a model as consisting of several graphs. There is no support for complex properties, effectively ruling out true modelling of object-oriented methods, and complex object support is limited to a simple explosion construct.

Despite these limitations, MetaEdit has proved successful as a metaCASE and CASE tool, with users numbering in the thousands. It has also been used as a tool in other research projects, e.g. Jarzabek and Ling

(Jar96) found it useful in developing a BPR method with CASE tool support.

MethodMaker (Mark V)

ObjectMaker is a multi-method CASE tool. Method support for it can be developed with the Tool Development Kit (TDK, described later) or MethodMaker. MethodMaker allows the "rapid development of new method notations" and to "modify and extend existing notations". Metamodelling is accomplished by with "with simple diagrams and fill in forms" by "editing a diagram set". The "ObjectMaker CASE Tool functionality is inherited": however, it is unclear whether this functionality is inherited into the metamodelling tool (i.e. ObjectMaker with a special metamodelling method) or the resulting tool (i.e. there is no need to specify tool behaviour in any kind of code).

Thus it would appear that metamodelling with MethodMaker follows the same pattern as MetaEdit, described above. Sadly, Mark V have declined to provide any material or details on their products, citing 'trade secrets'.

System Architect (Popkin)

Popkin's System Architect provides only limited metaCASE functionality: according to their web page[1] the user can "Define properties for any dictionary entry, including definitions, symbols and diagrams. Build links between various dictionary objects.". Definition of entirely new methods appears not to be possible. Again, attempts to obtain further information have been unsuccessful.

ToolBuilder (Sunderland/IPSYS/Lincoln)

The ToolBuilder metaCASE system was originally reported in (Ald91) and has since been commercialised. It currently consists of three components:

- the specification component — used to create the specification of the tool.
- the generation component — used to transform the specification into parameters for the generic tool
- the run-time component — the generic CASE tool itself

The first two are contained in the METHS system, and the third is called DEASEL. DEASEL provides standard CASE functionality and supports multiple users on a true repository. METHS captures four kinds of information:

[1] http://www.popkin.com/prods/sa/add.htm

- the data model upon which data capture and output generation is based.
- the frame model upon which the views are based
- the diagrammatic notation for each diagram frame
- the textual presentation for each structured text frame

The data model is ER extended with some constraints and the ability to have attributes whose values are derived from other attributes. It allows triggers on events applying to attributes and relationships. The frame model specifies editor behaviour, but DEASEL does supply a default set of behaviour.

A guidebook to using ToolBuilder exists on the WWW[2]. METHS definitions are in the form of textual files in three languages: data description language (DDL), graphics description language (GDL), and FDL (presumably frames description language). No information on the textual presentation language was available. The DDL and GDL appear not dissimilar to the corresponding parts in MetaEdit's textual metamodels, although the syntax seems somewhat harder to follow.

Thus ToolBuilder appears to provide a usable metaCASE system, but the time required to build support for a method is long:

"Even a completely new method can take only man-months to implement, with prototypes taking man-weeks. The rapid prototyping nature of ToolBuilder means that demonstrations can be created in man-days."

This is probably because of the complicated textual languages, and the separation of the CASE and metaCASE tools. Also, whilst it is of course a benefit that DEASEL provides basic default CASE behaviour, one gets the impression that this may be insufficient for most actual CASE tools, requiring coding in FDL to specify tool operations.

In comparison with MetaEdit, ToolBuilder seems to provide better support for large integrated models, and extra possibilities for customising the behaviour of the resulting CASE tool, but at a high cost in terms of the time required to develop CASE tools. Also, no information is provided as to if and how existing models can be updated in any way to reflect changes in the metamodels, a vital requirement for method engineering.

[2] http://osiris.sunderland.ac.uk/rif/bgtbk/wmwork/www/tbk_1.html

Research prototypes

ConceptBase (Passau/Aachen)

ConceptBase[3] is primarily a multi-user deductive object manager on top of a Telos database with extensions from deductive and object-oriented languages. As such a repository it has been used at over 150 educational institution sites. It is possible to use ConceptBase as a metaCASE tool, however most usage has been in other areas, with close examples of these being process modelling, design rationale, and retrieval of reusable software components. Some implementations of simple single methods with ConceptBase exist: ERA, a BPR method with four simple object types, and FUSION.

The conceptual support of ConceptBase is high, but its representational support appears low: symbols appear to be limited to displaying a single simple property (the 'label' or 'name'). More seriously, representation information of CASE diagrams is apparently not stored in the repository, but rather in separate files: no information is given as to how the problems this creates are solved in a multi-user environment. Further, loading such a file only succeeds if all the conceptual objects present when saving are still present when loading[4]. A detailed list of problems encountered with ConceptBase 4.0 — mainly its representational support — is given in (Hah96), which attempted to provide CASE support for FUSION with ConceptBase.

Metamodelling in ConceptBase appears to be partly textual and partly form-based. A drawback is the requirement that one property of each object, its 'label' or 'name', must have unique values. Whilst this was common in older structured methods, it is not necessarily the case in newer methods.

Whilst the repository is generally the strong side of ConceptBase, it has two serious drawbacks when considering serious use as a metaCASE tool: it is required that the whole repository must fit into main memory, and there is currently "only primitive provisions for multi-user support".

However, whilst these problems are serious, and render ConceptBase at best only a research prototype for metaCASE, it must be remembered that ConceptBase is not attempting to be specifically a metaCASE tool. Its strength lies in its deductive temporal repository which allows it to be used beneficially for a wide range of research, and in that it excels.

---

[3] http://www-i5.informatik.rwth-aachen.de/CBdoc/cblit.html lists a good selection of references, many important ones of which are only available as downloadable technical reports.

[4] http://www-i5.informatik.rwth-aachen.de/CBdoc/userManual/ see section on Graph Browser.

KOGGE (Koblenz)

KOGGE (Ebe97) is a metaCASE environment that follows a similar transform-compile-link paradigm to MetaEdit. The same basic functionality of the CASE tool is present whether metamodelling or modelling, but these two functions cannot be carried out simultaneously. KOGGE uses an extended ER meta-metamodel for the basic concepts of methods, which are modelled with a graphical editor, and this is extended by the GRAL language for describing constraints and rules on the concepts. The operational user interface is described by separate state charts, also modelled graphically. In addition, users must program part of the CASE tool functionality in a language similar to Modula-2. Although there is a library of routines for some often used actions, the need to right Modula-2 programs as text means that KOGGE does not fully meet the criteria we laid down for a metaCASE tool.

MetaGen (Paris)

A prototype offering only basic CASE support, MetaGen (Rev95) is interesting because of its support for transforming a model from one method to another, via a set of production rules. Methods are modelled in either a graphical or a textual list-based user interface, and then transformed to Smalltalk classes. These classes form the metamodels, and are instantiated when modelling. In addition, rules must be declared in a separate system, NéOpus, and the task of writing the rule base is a 'fully-fledged problem'. When the rule base has been written, the transformation can be applied to the source metamodel to generate the target metamodel, as well as later on source models to generate target models: a clever touch. Basic facilities exist to export and import metamodels as text.

MetaPlex

Although MetaPlex (Che89) had basically textual rather than graphical CASE support, it is worthy of mention as one of the earliest metaCASE tools. It used a textual language to define metamodels, which are interpreted rather than compiled, and even included some rudimentary CAME functionality by generating help text for method users from the metamodel.

MetaView (Alberta)

Initially described in (Sor88), the MetaView system has been extended to include textual metamodelling with extended ER and graphical definition of symbols. Support as a CASE tool is rather weak, with only a simple graphical editor which lacks some standard functions. The result is rather like an early stage in the history of MetaEdit without graphical metamodelling, indeed a Master's thesis (Lo95) that produced a partial prototype symbol editor states that:

"We are most interested in MetaEdit's graphical meta-modeling techniques because it is the ultimate goal of our research in the Metaview system. Since the general architecture and the approaches used by MetaEdit are quite similar to those of our Metaview system we have used the MetaEdit system as a valuable source of reference during our research."

Insofar as this represents a willingness to build on existing research efforts, it is a rare and welcome sign of the use of constructive research to its full potential. MetaView includes interesting ideas, in particular to do with transformations between models made with different methods, and we await the fully working version of those parts with interest.

RAMATIC (SISU)

RAMATIC (Ber89) is a prototype metaCASE tool supporting textual definition of the conceptual constructs of methods, their symbols, forms for editing them, and the menu and toolbar structure of tools. Menu items can be defined to call existing functions, or extra functions can be added by programming: the most basic CASE functionality is already provided in the existing set of over 100 functions. An important feature in RAMATIC is its separation of conceptual information from representational information. RAMATIC is compared with QuickSpec and Excelerator in (Mar93).

However, the time to create a CASE tool for even simple methods such as ER and DFD is of the order of weeks. Partially to address this, an interface from MetaEdit (Ros92) was built to enable graphical metamodels to be transformed into the textual language for conceptual constructs used by RAMATIC. With extensions to include the menus and symbols (not built), such an approach was estimated to reduce the time to build a CASE tool by an order of magnitude, from weeks to days.

Unavailable products

The following products are currently not available: manufacturers are however already advertising their functionality, or a previous version has been available and withdrawn. Only especially interesting examples of such products are included here.

Customizer/Excelerator (Index Technology/Intersolv/SELECT)

The current status of this tool is rather uncertain: Customizer was a tool that allowed Excelerator CASE tool users to change the metamodel used in their repository. Excelerator II has now been transferred to SELECT Software Tools, who intend that customers should stop using it and move from it to their own CASE tool, SELECT Component Factory, which has no metaCASE features. Attempts to contact the manufacturers have met with no success.

According to an old FAQ[5], Customizer offered the following functionality:

"Can customize and mix parts of one approach with another in a user-friendly manner. Intersolv LAN repository meta-CASE [tool] with customizable graphics and rules"

Whilst earlier versions of Customizer used a procedural language for checking of consistency constraints, later versions used the CLIPS rule-based language acquired from NASA (Hag95). Prior to this, Customizer probably could not be classified as a metaCASE tool because of the programming required, but it had been used by Tagg (Tag90) to implement support for a small method. The main benefit identified compared to other methods of building CASE tools was the short time required to make changes, allowing comparison of different versions or implementations of the method. Similarly, DMR Group, a systems integrator based in Montreal, added support for their Productivity+ method using Customizer (Han94). Customizer was compared with QuickSpec and RAMATIC in (Mar93), which also provides more details of its metamodelling facilities, albeit before the change to use CLIPS.

GD Method Builder (ASTI)

This was originally advertised on their web page as being available '4th Quarter 1997', but the date has now been removed. The release date for the accompanying fixed-method CASE tool, GDPro, has been put back to '3rd Quarter 1997'. The sales material on the web advertises an 'easy to use GUI' to define new objects, relationships, attributes and symbols and code generation for these. As they put it: 'No more struggling with text files and complex, abstract description languages. Welcome to the age of GUI-based meta-CASE'. We await the finished product with interest.

The CASE tool, GDPro, will support 'collaborative "white board" development', allowing users 'to interact "live" with each other': an interesting innovation, although one whose usefulness in practice is dubious (Mar91). Such an approach was also taken in Nokia's TDE (Tai97).

Paradigm Plus (Protosoft/Platinum)

Paradigm Plus is a multi-method CASE tool, whose method support is developed using the Paradigm Plus CASE Developer's Kit (CDK). This CDK has often been advertised as being available for purchase, which would make Paradigm Plus a metaCASE tool. However, whilst the CDK has at some points been available and actually sold, at most points it has not actually been available: this is the case at present. Currently, Paradigm Plus's customisability is limited to a high-level script language that

[5] http://www.cyberdyne-object-sys.com/oofaq/ comp.object FAQ, v1.0.9, 4/2/96 Appendix D

provides access to the data in the repository and some functions of the tool[6]. It does not however allow customisation of methods: this is 'coming soon' (Keu97). Sadly, no description of what the CDK is, what it does or how it works is available.

Earlier user reports seem to have been mostly negative. A project at the University of Iowa in Spring 1996[7] gave the following description of the CDK:

"One major drawback of the current Paradigm Plus system is that it is not easily extensible to support other methodologies, nor is it easy to update currently-supported methodologies. Current estimates for such changes provided by ProtoSoft are: sixteen person-months for a new methodology to be completely incorporated into Paradigm Plus, and seven person-months for an update to a currently-supported methodology."

The project was apparently given requirement specifications by ProtoSoft, who wanted them to build a form-based GUI (similar to that in MetaEdit+) for metamodelling with Paradigm Plus. Such a GUI was implemented by the project, but the resulting 'Paradigm Plus Plus' has not been commercially released, tested or further developed. Our own experience with form-based GUIs for metamodelling would indicate that it would noticeably reduce the long times for method development quoted above.

VSF (Systematica/ISDE Metasoft)

VSF (Poc91) used the set-theoretical and propositional calculus language CANTOR to define the conceptual data in metamodels and its constraints, and the Graphical and Design Language GDL to specify graphical representations. The latter was somewhat complicated: 15 lines of code to represent a simple DFD Flow arrow. Despite this complexity, VSF was taken into use to some extent (e.g. as the basis for the MEET environment (Hey93a, Hey93b)). VSF's strong point was its ability to define complicated constraints, although other research (Ves92) suggests that these are not generally appreciated by users. A clear weakness is the complicated nature of metamodelling: the time to metamodel a method would be considerably longer than with today's leading tools.

Although ISDE Metasoft offers support for existing customers, VSF is no longer being sold.

---

[6] http://www.platinum.com/clrlake/para_30/features/custom.htm

[7] Project members were Pramada Boinepalli, Dave Frank, and Julie Jones. See http://www.cs.uiowa.edu/~frank/softeng/

Non-metaCASE tools

A metaCASE tool, as we see it, is more than just a system which allows the development of CASE tools: if that were the case, any programming language would qualify as a metaCASE tool. We thus exclude tools which require programming (in something like a standard programming language) to specify basic tool behaviour, although we allow specification of meta-datamodels and symbol definitions in a textual language, and also some measure of specification of constraints in some kind of logic language. Basic CASE tool functionality should be provided by the metaCASE tool, and modified by the data in the metamodel provided by the metamodeller.

Similarly, resulting CASE tools must be CASE tools, not for example databases or flowcharters. There is of course some measure of overlap and fuzziness in these definitions, and not everybody would accept all the criteria, and it is for this reason that the following are at least briefly described, rather than left out completely. In addition, by describing tools just outside the boundary we set for metaCASE, the positioning of that boundary hopefully becomes clearer to the reader. Where there are several examples of a certain type of tool, only one has generally been included here. As should be clear, the inclusion of a tool here is no slight on its usefulness or value: in the right situation each of them has its use, and research value in its own field.

Decamerone (Twente)

Harmsen et al. (Har93, Har94, Har97) have described a prototype CAME environment, Decamerone, which assists the method engineer in selecting among fragments from several methods to assemble a method suitable for his current contingency. The resulting set of fragments is processed and can be fed into the Maestro II repository to generate the basic repository definition for a CASE tool supporting the new method. However, it was necessary to write source code in a programming language to specify the actual CASE tool functionality for the chosen situational method (Har97, p.274), and thus does not qualify as a metaCASE tool by our definition. The prototype implementation encountered a number of problems, attributed to the lack of suitability of Maestro II as a platform for metaCASE.

GOODSTEP (Frankfurt)

The GOODSTEP project (GOO95) extended the O_2 ODBMS and built a framework around it for building CASE tools. The GraphProject framework component can generate graphical CASE tools from concise GSTL pseudo-code, and indeed such a tool has been generated that can transform a diagram into an O_2 schema, although this has not apparently been followed up. 12,000 lines of GSTL code were needed to develop a CASE tool for British Airways (Emm97); whilst this amount of code places it outside the realm of true metaCASE, the fact that the code expanded to

215,000 lines of C++ shows that the approach has benefits compared to coding a CASE tool from scratch. Overall, GOODSTEP represents more of a repository and framework than a metaCASE environment, like other similar tools such as Ptech. As a repository, and thus in the completed CASE tools, strong features are its high concurrency, small granularity of locking and version management.

GraphTalk (Rank Xerox/Parallax Technologies/Socs)

GraphTalk appears to follow a similar transformational metamodelling paradigm to MetaEdit. However, in addition to drawing a graphical metamodel the user is required to specify update operations and checks in the GraphTalk QueryLanguage (GQL), which is similar to SQL, and this is all then used coupled with C++ to generate source code. No further information could be found to confirm these statements from a brochure and a user: the product appears to have changed hands several times, and it appears not to be currently available.

Hardy (Edinburgh)

Hardy is mentioned in several places as a metaCASE tool, but it is primarily a graphical diagramming tool. It thus can provide good support for drawing diagrams conforming to various methods, but little support for storing the conceptual data essential to CASE.

Maestro II (Softlab)

Maestro II (Mer91) has been used as the basis in the prototype Decamerone CAME environment (Har97). It is more of a repository than a metaCASE tool, and in addition to specifying the metamodel the user must write code for e.g. storing conceptual data and performing graphical operations (Bri95, p.27–32).

MetaDesign (Meta Software Corp.)

As their web page puts it, "MetaDesign is an easy-to-use flowcharting tool[8]". It has however sometimes been erroneously mentioned as a metaCASE tool. It has none of the other prerequisites of a CASE tool, such as different property types, code generation etc.

MViews (Waikato, NZ)

MViews (Gru96b) is a framework for building CASE tools using the CoCoA meta-metamodel. Currently, the implementation requires hand-coding repository and tool information from the CoCoA models, but a prototype is under construction to automate this process, although the aim

[8] http://www.metasoftware.com/prodmdesign.htm

is not to totally avoid coding in building a CASE tool. The interesting features of MViews are that it has other prototype implementations which variously provide CSCW support for CASE (Gru96a) and automatic continuous maintenance of a parallel model in different methods (Gru95). We await the integration of these research threads into a single functioning environment with interest.

ObjectMaker TDK (Mark V)

ObjectMaker is a multi-method CASE tool. Method support for it can be developed with the Tool Development Kit (TDK) or MethodMaker (described above). The TDK allows customisation of methods by providing "direct access to the predicate logic rules that create, control, and pre- and post-condition all tool behavior". Because the metamodelling process requires writing code to specify tool behaviour, we do not count this as true metaCASE.

SUMMARY

Examining a large number of tools has given us an insight into the various ways that different parts of metaCASE functionality can be implemented. In general, a metaCASE tool provides a way to describe a method, which can be broken down into its static concepts, the constraints that govern how these may be linked and used, and the symbols used in the graphical representation of the method. In addition, some metaCASE tools require specification of some of the functionality of the final CASE tool, whereas others generate this automatically on the basis of the method.

For each part of the method description above, there is some kind of language. The language may be programming language code, a logic language, a textual language, or a graphical language. Where a graphical language is used, it is normally first transformed to a textual language, but this need not concern us: the metamodeller may work entirely with the graphical language. Similarly, the textual language is often internally turned into code and compiled to drive the CASE tool. Sometimes, textual languages are transformed into logic languages used directly by the CASE tool; a logic language can also be transformed into code. This gives us a good ordering on the ease of use of these different types of languages: if language A for the metamodeller is transformed to language B by the metaCASE tool, language A must be easier to use. This is born out by practice: a graphical metamodelling language is easiest, then a textual language, then a logic language, and finally code.

The table below takes the most important metaCASE tools examined, and shows which kind of language they use for each part of metamodelling, including specifying the CASE tool functionality if this is not fully automatic. The manufacturer's estimate for the time in man-days required to metamodel an average-sized method, e.g. Structured Analysis, is also given: where no such estimate is available I include my own view with a question mark. My own

estimates are conservative: recall addition of a method into Paradigm Plus was estimated by the manufacturer to take over a year.

Tool	Concepts	Constraints	Symbols	Functionality	Days/method
MetaEdit Personal	graphical	graphical	graphical	automatic	3
KOGGE	graphical	logic	textual	graphical+code	10?
MetaView	textual	code	graphical	automatic?	10?
ToolBuilder	textual	textual	textual	textual	40
VSF	textual	code	code	code	80?

As can be seen, the times differ widely. The times are also a measure of the complexity of metamodelling with that tool. Higher complexity implies more time for the same method, and also a greater chance of problems later on, if metamodelling follows the same pattern as other modelling efforts. However, higher complexity could also allow more precise metamodels and thus better support of the method. Taking this to an extreme, making a basic CASE tool purely by programming would at the least require a man-year, and to take advantage of the complexity of programming to provide more precise method support would turn that into several man-years. Thus it would appear that the benefits of complexity are outweighed by its problems: to achieve the same basic results that can be achieved in a less complex language takes far longer, and to use the complexity to improve on the results of the less complex language takes much longer still.

At present, there are no quantitative empirical results as to how much the more complex languages increase the time required for a basic metamodel, or how much the extra precision they can offer helps the CASE tool end-user. It appears though that the effect on the time is considerable, and empirical research indicates that there is little point expressing fine details in a metamodel. A study of expert users of the same method (Hof96) revealed that the implicit 'personal metamodels' of the method they followed differed to a considerable extent, even at a fairly high level of granularity (above the level of manipulating individual objects). The authors concluded that it was not desirable to capture the fine details of one user's implicit metamodel, but rather that more important was to include the main details, and allow variation. This would imply a small rather than large set of constraints (allowing variation on the model level), and the ability to change the metamodel on the fly, with models automatically updating (variation on the metamodel level).

Thus whilst a metaCASE tool must be powerful enough to allow modelling of all the conceptual structures of a method, the facilities for expressing constraints need not be made so powerful, especially not where the increase in power would mean more work to metamodel methods simply.

3.2 Problems

From this basic overview of the situation and descriptions of existing tools, we can now move to examining more specifically the problems and issues that have been identified with CASE, metaCASE and CAME. We divide the

problems into five areas: poor success of CASE, methods changing when applied, weak method integration in CASE, metaCASE needing extension, and immaturity of CAME. For each problem area, we critically examine relevant literature and products, giving a précis description and conclusions. The title of each problem area identifies the problem and gives a short summarising comment (in italics) on the conclusions we draw. At the end of each area we bring together the conclusions from the material examined into a short paragraph (in italics).

1: CASE tools have shown only fractional success…
both organisational and technical problems, CASE very important in future

It is now widely recognised that CASE tools, although in general helping in the situations where they are applied, have failed to have the expected impact on the market.

In a survey, ISD managers were found by Norman and Nunamaker (Nor89) to recognise the benefits of CASE as providing improvements in quality, maintainability, and productivity. Siltanen (Sil90) showed that these improvements were made possible by a strategic orientation in the company. Correspondingly, Parkinson (Par90) found that CASE is not taken into use effectively if there is no existing methodology in use in the company: if there is, then CASE can make enough of a difference on the productivity side that resources can be released to focus on quality.

The basic use principles of CASE have remained unchanged from the early days: for instance, Charette (Cha86) describes how a software engineering environment could be useful at the various stages of software development, identifying principles that still form the basis of CASE use today. Similarly, some of the problems remain the same, and will probably always require more work: McClure (McC89) lists user-friendliness, intelligent method support, and reuse support, all of which are still issues that users cite today.

One of the main tool-centred problems in CASE is the method support. In a survey study of CASE tool selection (Sav93), the most highly emphasised decision factor was the 'range of application area of the tool'. This range can be improved in two ways: vertically by adding support for methods in different phases of ISD, and horizontally by adding alternative methods for different contingencies in the same phase of ISD. Equal first with that criterion were 'compatibility with other tools and methods', emphasising a broad and non-partisan method support, and 'support of design methods in use', again motivating a wide method base, but also pointing towards method integration, and configurability and flexibility of method support.

These findings tie in with those of other empirical researchers, who agree that fixed method CASE tools are of limited use in real-life situations such as information systems planning (Bri90, Wij91, Ste93). However, in spite of these problems, Kusters and Wijers (Kus93) found that 89% of ISD managers see the future importance of CASE in their organisation increasing.

Thus CASE has shown that it can offer substantial benefits, and the market seems committed to its use. Current CASE effectiveness is however plagued by organisational problems and problems in the tools, notably the lack of support for the user, and the inflexible, unintegrated method support.

2: Methods in use are different from defined methods…
and thus we need flexible, modifiable method support in CASE tools

As recently as 1988, Chikofsky and Rubinstein (Chi88) could say that actual method use in organisations was rare. Whilst still not the norm, there are now sufficient organisations using methods regularly for investigations to have been carried out into the nature of how a method is taken into use in an organisation. Aaen et al. (Aae92a), in a study of CASE and method use in the Netherlands and Finland, found that a significant minority (21–45%) of organisations needed changes to the methods and tools before they were used. Correspondingly, Aalto (Aal93) found that in actual use Rumbaugh's OMT method (Rum91) differed from that specified, being adapted to the local situation, thus motivating flexible method support in CASE tools. Ter Hofstede and Verhoef studied expert users of a method (Hof96), and found that there were significant variations both between experts, and also with the same expert at different times. Stobart et al. (Sto93), combining the work of four research groups in three countries, found a major criticism to be that CASE tools often required a change to methods other than those used in the organisation.

In a survey of how CASE tools are taken into use Aaen (Aae92b) found that tools must support existing work routines, and offer immediate advantages. Otherwise the bootstrapping fails before any long term advantages can be realised. Similarly, Urwiler et al. (Urw95) tested 70 users to find factors influencing the success of a specific CASE tool's introduction. Existing use of the method supported by the tool was the most significant factor influencing perceived quality improvements.

The degree to which the user of a method is allowed to deviate from it varies between methods, and also between organisations. The more prescriptive a method or organisation is, the less it will allow individual variation, and the more suited it is to the application of fixed method CASE tools. Even within fixed method tools, there is some scope for variation: Vessey et al. (Ves92) examine 12 such CASE tools, and identify three different philosophies in how strictly the tools implement the constraints from the methodologies. Restrictive tools do not allow any deviation, giving an error message at the time the user attempts to deviate from the method; guided tools give warnings either at the time of the action or when exiting from the editor; and flexible tools give warnings when exiting.

Of all areas of information systems modelling, information systems planning (business systems planning, business process re-engineering etc.) has perhaps the most prescriptive methods. Huge manuals list (supposedly) every model and item of data that should be produced, and give detailed instructions for how to do this. Even here, however, Stegwee and van Waes (Ste93) conclude "the next generation of CASE tools for ISP should be customisable". Interestingly, they rule out the possibility of a separate metamodelling environment producing a metamodel or CASE shell, saying "the tool itself should provide facilities for introducing new modelling techniques… by modifications in the metamodel of the tool". They thus favour an integrated metaCASE and CAME environment.

Cronholm and Goldkuhl (Cro94) analysed the kinds of method customisations that take place, and the motives for them. They found the main reason for choosing a metaCASE tool for normal CASE was that there existed no existing tool that supported the method they were already using, and programming a CASE tool was prohibitively expensive. As we have mentioned, methods greatly outnumber tools, thus this is in theory true for almost all organisations. However, there is often a tool that supports a method close to the one in use, and many organisations therefore make do with the reduced tool support which is available. This path is more dangerous than is realised, and often embarked on too lightly: Le Quesne (LeQ90) shows that changes in working practices and systems development methods should be evolutionary when introducing CASE, rather than trying to force an existing method in use to fit a tool, or a fixed tool to fit an existing method.

In the choice of metaCASE tool, Cronholm and Goldkuhl found, in addition to normal factors for buying a product, that the degree of adaptation possible and the presence of support for some parts of the actual method in an existing customisation were common motives. The time required to develop support for the new method varied from three to thirty months, depending on the complexity of the method and resources allocated. Some projects ceased after only some of the goals were met, because of lack of resources.

So far, we have worked on the tacit assumption that, although a method-in-use may differ from the method on paper, at least the method on paper will remain the same. There is no doubt, though, that methods evolve (Bubenko and Wangler (Bub92 p393) list examples) and an organisation's ISs need to evolve quickly to work in a rapidly changing and complex information arena (Ste93).

Whilst ISD methods are useful, in practice no one method is sufficient for all situations, and a selection of a method suitable for the contingencies must be made. Even then, this method will be altered in actual use. CASE must support the methods already in place in an organisation, and allow them to change as the project evolves.

3: CASE support for multiple, integrated methods is weak…
new CASE tools must provide better linking between methods and their models, in a way the user chooses

The weakest part of CASE tools today is their ability to support method integration. Although a CASE tool may support multiple techniques or diagram types, the linking of data in a graph of one type to that in a graph of another type is poor.

Current CASE tools often provide only one linking mechanism between graphs, namely explosion. This links an object in one graph with another graph, so that the second graph can be easily opened starting from the object. The actual functionality and semantics of this link varies, but it is rarely sufficient in functionality or expressive power to allow flexible, interface-based reuse, in a manner similar to CAD or chip design. In this respect it is interesting to note that Savolainen (Sav93), in a survey of CASE in use, finds that 'Reusability of descriptions' was one among four advantages of the use of CASE tools that rose as a group above all the other reasons. Similarly, Banker and Kauffman (Ban91) studied 20 projects in the banking sector, and found an order of magnitude improvement after the introduction of CASE, with reuse an important driving factor. Thus, reuse support is an area where CASE is already showing success, and effort on improving the functionality there is well-placed.

Hochstettler found that in practical use information is seldom updated retrospectively into methods from an earlier ISD phase (Hoc86 p.17), thus leaving the data in the repository semantically inconsistent.

Lehman and Turski (Leh87) emphasise the need for method integration, in addition to flexible methods. Similarly, Goldstein (Gol90) finds that tools should be capable of 'learning' the methods required, which should be integrated so that information can be shared between them.

Goldkuhl and Cronholm (Gol93), in a review of metaCASE tools, find insufficient support for method integration on the type level, which is also reflected on the instance level by poor user interfaces which prevent work in multiple methods.

From a large survey in three countries, Stobart et al. (Sto93) present main criticisms of CASE tools, which are stifling their adoption and use. First among these are the lack of the possibility to use several methods in an integrated manner, and the poor integration between tools for different software development phases. Tools are criticised for forcing a change to methods other than those already used in the organisation. They also find that the integration of the visual representation with the conceptual repository data is weak, and that there are no multi-user facilities.

A survey of non-use of CASE in Slovenia (Rup95) found the most cited problem to be lack of integration, both within a CASE tool and with other existing tools and procedures. Combining their results with a survey of CASE users in the UK, they find the strongest future requirement is for better integration with programming via code generation and reverse engineering, followed by multi-user functionality, method aspects, and customisability.

Brinkkemper (Bri93) provides us with a framework for analysing method integration (he calls it modelling transparency). He gives four levels:

0. Stand-alone CASE tools
1. Single window, shared repository
   Links can be made between diagrams, but to follow a link the user must close the current diagram
2. Multi-window, shared repository, fixed limited range of link types
   Updates in one window are reflected immediately in another, but links can only be created in certain situations
3. Multi-window, shared repository, much freedom to create links
   Links can be created at will, in a manner similar to hypertext (Smi88). This functionality is called Hyper-CASE by Cybulski and Reed (Cyb92).

Brinkkemper states that most current CASE tools support level 1, and better support would be "an important functionality to increase the productivity of CASE tools." He goes further to say that a high degree of method integration "is one of the absolute requirements for advanced systems development".

Further evidence for the importance of a natural user interface is provided by Chau's survey of around 100 CASE users (Cha96), which found that ease of use was the strongest determining factor on acceptance (0.44, where 1 totally determines and -1 totally inversely determines), with considerably more impact than organisational factors (transitional support and perceived short and long term benefits).

Evidence of the increasing importance of CASE was found by Kusters and Wijers' survey (Kus93), in which 89% saw the future importance of CASE in their organisation increasing. The three top selection criteria for CASE confirm our findings above: degree of coverage of techniques already in use, degree of integration between techniques, and quality of man-machine interface.

CASE tool support for the integration of different parts of a method is poor, severely weakening their applicability as tools to support the whole life cycle of an ISD project. Better facilities are needed to support reuse and other linking between graphs, and to better integrate information in multiple windows via common conceptual data in the repository.

4: MetaCASE is useful but needs extending…
especially its data models, user-friendliness, and method integration

Hochstettler (Hoc86) states that a datamodel (metametamodel) needs an easy computer implementation, making it easy to use to define methods. He found that relations are not good at representing methods, and models that use graphs are better. Attribute grammars were good (cf. Sae94) but too complicated to use, and definitions were unnecessarily large. He created the TRIAD metametamodel which used a combination of these, and implemented it for proof of concept. His system had only poor support for intergraph links, and only two kinds of relationships, one for hierarchical and one for 'other' kinds of

relationships. The definition of the graphical aspects were mixed in with the conceptual, assuming only a single representation of each concept. MetaCASE in his tool took place on a low level: the user had to write his own interfaces and extended commands, and there was no interactive editing. The metamodelling was carried out via a primitive command-based interface.

Tagg (Tag90) points out that, while there are numerous papers on CASE technology, methodologies, and successful application of methodologies in software design, there are but few on customising CASE environments to support new methodologies. He used Customizer (Exc87) to model the Box Structures Method. For him, the most important result was the validation of the strategy of using a tool customiser. The time to create different versions was minimal, providing the opportunity to choose the best implementation. Whilst these results can rightfully be regarded as most encouraging for metaCASE, they need balancing with other work with more complicated methods (BSM has only seven components, two relationships, and one graph, plus two other graphs that are restrictions of the first).

Amundsen and Christoffersen (Amu87) used Excelerator and Software Through Pictures to model the SUSI method. They found these metaCASE tools had good support for modelling individual components, but only poor support of links between graphs, i.e. method integration.

IBM chose to use Systematica's Virtual Software Factory metaCASE tool (Poc91) in building its BSDM support tool. Haine points out the economic justification of this: "building a toolset from scratch would have taken longer, [and] been very much more costly" (Hai92, p.140). He estimates that by using metaCASE the user ends up with an organisation wide licence for a full life-cycle toolset for less than the cost of equipping 20 developers with some standard fixed-method CASE tool. With other metaCASE tools, the situation could have been even better: VSF cost around £100,000 and was thus probably the most expensive metaCASE tool available.

Jarzabek and Ling (Jar96) used MetaEdit in the development of a BPR method and its implementation. They note the usefulness of the metaCASE environment in development, as the method and tool can be easily changed as required, allowing experimentation in a cost-effective manner. Whilst praising the 'powerful report generating facilities', they found the lack of a unified repository for all models a problem, and would require a platform independent solution for further work.

Goldkuhl and Cronholm present a framework for the design and evaluation of metaCASE environments (Gol93), and identify problems with the ease of use of their metametamodels, the power of the these models to accurately describe the concepts and representations of methods, and their ability to integrate methods.

Tolvanen et al. (Tol96) surveyed existing research on method engineering, categorising it according to research subject (technology, language or organisation) and research paradigm (survey, field study, laboratory experiment, case study, action research, applied, basic or normative). They found that several areas were significantly over-populated, while others showed no current research. In particular, the emphasis was on constructing

new tools and theories, rather than evaluating existing ones. Whilst they found this emphasis to be expected of a new field, with few possibilities for real-world empirical observation, the total lack of any research in the laboratory experiment paradigm (the only totally empty paradigm) appears worrying.

MetaCASE has been demonstrated to be effective, but current tools' metametamodels are often hard to work with, and insufficiently powerful to allow accurate construction of an integrated set of methods. Empirical research is needed, albeit only laboratory experiments, to justify claims, examine the efficacy of existing approaches and explore the feasibility of new approaches.

5: CAME a largely unknown area, where research is only just starting…
need better support for method engineers, better linking of CAME to metaCASE

In the seminal paper on method engineering (Kum92: an earlier version was published in 1988), Kumar and Welke present it as a systematic discipline akin to software engineering. They stress the importance of reuse of method components, and of the analysis of the results of the engineering by observation of methods in use.

Tolvanen and Lyytinen (Tol93) also stress the importance of a systematic approach to metamodelling and method development, and present a method or meta-method for carrying this out. This is further developed, with more emphasis on the method engineering aspects, especially how to carry it out incrementally, in a later paper (Tol95).

Cronholm and Goldkuhl (Cro94), in a survey of metaCASE in use, find no evidence of a meta-method in use.

Part of method engineering is the ability to analyse a methods into its components (metamodelling) and to compare these analysed methods. Brinkkemper et al. (Bri89) use this to establish 'method companionship', the way a CASE tool works with a method. Oei and Falkenberg (Oei94) use metamodelling to establish hierarchies among sets of related methods, allowing more informed choice of method for a given contingency (Kot84). Similarly, Rossi and Tolvanen (Ros94a) metamodel methods and use the resulting metamodels as a better-defined, more objective basis for comparison of the methods.

Harmsen et al. (Har93, Har94, Har97) have described a prototype CAME tool, Decamerone, which assists the method engineer in selecting among fragments from several methods to assemble a method suitable for his current contingency. The fragments include information about their semantics, likely range of application, and their description in the metalanguage of Maestro II. Despite problems with the prototype, it demonstrated the beneficial possibilities of an environment integrating metaCASE and CAME.

Harmsen et al. (Har94) also examined other existing CAME tools, such as Anderson Consulting's Solution Configuration Tool (Hid93) and Ernst and Young's Navigator system (Ern93). They criticise these as they allow selection of fragments from only one method, reducing the possibilities for configuration and the likelihood of a good solution to the current contingency. Moreover,

they lack customisable CASE functionality (Har97, p. 246), thus methods cannot be tested as they are developed.

Heym and Österle made use of VSF (Poc91) in the construction of their MEET method engineering environment (Hey93b, Hey93a): an interesting use of metaCASE to build, not a CASE tool, but a CAME tool. The actual support available for CAME is limited: the inherent low-level metamodelling of VSF makes the tool more like a metamodelling aid than an environment where methods could be systematically engineered and tested.

Both Decamerone and MEET, probably the two most notable attempts at broad CAME support with metaCASE, met problems because of the low level of metamodelling (in practice coding) required in the underlying systems they used for metaCASE. This provides a strong indication that a powerful metaCASE tool with metamodelling on a high level would be a critical success factor for a CAME tool.

There is a significant gap between CAME and metaCASE that needs filling: CAME tools currently do not support metaCASE (modelling methods and supporting them in use), while metaCASE tools work on too low a level, providing little support for the process of developing and testing methods. An environment that linked CAME and metaCASE would improve the efficiency of both, and empower still better studies of method classification and contingency factors.

6: Tool support for CASE and CAME largely single user…
need integrated support for multiple methods in multi-user, repository based tools

CASE technology was originally intended as support for single developers in designing and documenting their part of the system. The current direction and need for the CASE tools of the future is support for multiple users and multiple methods. Stobart et al. (Sto93) found the lack of multi-user facilities to be a major criticism of existing tools. Looking broadly at the field we can see that solutions for method integration are being promoted in two different paradigms, metaCASE and open CASE. MetaCASE proposes the creation of CASE tools which can be configurable for any method, whereas open CASE sees many fixed-method CASE tools communicating via common standards.

Currently, metaCASE is leading in terms of commercially available implementations, the support for open CASE having become stuck in the mire of unadopted standards and multiple conflicting standards. The nearest thing to open CASE so far is the ability of certain CASE tools to import information from certain other CASE tools. Commercial considerations prevent the implementation of similar export functions: manufacturers want to keep customers disinclined to move their data to the tool of another vendor. The integration of CASE tools thus remains on a static level: data is not shared dynamically, and updates are propagated manually, in only one direction.

One extension of this current solution of is presented by Papachristos and Gray (Pap94). Their idea, which they call federated CASE, is based on a central tool that would handle the conversions between the various CASE tools. For each CASE tool there must be an input filter, that would transform the data into

a common format, and an output filter, that could take data in the common format and turn it into the input format needed for that tool. A similar philosophy is seen in the attempts to create a common format for CASE data, called CDIF (CDI91). Despite the backing of major players, CDIF progress has been slow, and many areas remain undefined. In particular, the CDIF standard only allows transfer of models of a certain severely limited set of methods (basically Structured Analysis, with support for object-oriented methods under development). This effectively curtails its usefulness, as addition of a new method to CDIF requires several years of work. A further problem is the lack of implementations adhering to the CDIF specifications: CASE tools that claim to support CDIF actually support their own dialect, which may or may not be compatible with the dialect in other CASE tools.

Both federated CASE and CDIF offer only static data exchange, with all the ensuing problems of consistency. Similarly, they only really offer help for models in different CASE tools that follow the same methods: exchanging information between different methods on this basis is largely a semantic impossibility (Bub92, p.405). Further, working with a heterogeneous set of tools is highly unsatisfactory, as they are not integrated with, for instance, similar user interfaces: as Stegwee and van Waes (Ste93) quote, "A computerized design tool is a great help. Two computerized design tools are a disaster." These problems are multiplied significantly in the multi-user situation, for which these offer no help. Thus, although federated CASE and CDIF may have their uses in certain situations, we can rule them out as a serious answer to the overall problems in this field.

Whilst metaCASE and open CASE are often seen as competing solutions, Bosua and Brinkkemper (Bos94), in presenting a review of the current state of the field of CASE integration, point out that we need both better integration within environments (as promised by the next generation of metaCASE tools), and better integration between environments (as promised by open CASE).

Some standards have appeared for the architecture of open CASE tools, e.g. the ECMA PCTE (Tho89), which suggests multiple tools accessing common data in a shared repository, and presenting it to the user via a common user interface and display conventions. Similarly, standards for the repository, the heart of CASE data integration, have been suggested, but none of these appears to have taken off. PCTE included a description of the repository as well as the architecture, and suggested a move away from the widely-used SQL databases. SQL continued to be used, however, by IBM's repository manager for AD/Cycle, also proposed as a standard (IBM90), and as the basis of IRDS (ISO89), the ISO standard for CASE repositories. In an analysis of these standards, Olle (Oll92) concluded that "the inconsistencies among the three approaches threaten the ideal" of open CASE.

Whilst true open CASE implementations are not available, the theory of open CASE has two major advantages over most current metaCASE tools. Firstly, open CASE per se already provides support for multiple users accessing a common repository, whereas most metaCASE tools are single user even for CASE work. Secondly, open CASE already includes the idea of integrating multiple tools, possibly even with multiple representations of the same

conceptual information stored in the repository (Stobart et al. (Sto93) noted this need in their survey); current metaCASE tools have only one editing tool, for diagrams. For metaCASE to offer a viable alternative for large scale projects, it too must move to adding multi-user support, and multiple editing tools for different representation paradigms.

An important issue in either solution is the nature of the underlying repository. In order to provide adequate support for multiple users, reuse, and changes to methods this must be a true object-oriented repository, and not a file-based or relational storage system, as is common in current tools.

Thus, despite considerable effort to promote CASE environment integration, as far back as 1989 Luchner et al. (Luc89) could sum up the current situation, saying that tools can be integrated either by an exclusive mutual link between two tools, which is an optimised but tool-pair specific solution, or different tools could be connected by a common data model for integration, which is a general but very complex solution. The field must move towards the latter, more general solution, and metaCASE currently appears to provide the better chance of success.

Of course, as Bosua and Brinkkemper (Bos94) mentioned there is nothing to stop the integration of several metaCASE tools via open CASE standards, if and when such standards become apparent. The main problems remaining with this solution would be the integration of the different meta-metamodels used by the metaCASE tools, although this would be eased in many cases by the fact that most metaCASE tools use some extended ER variant as a meta-metamodel.

Need multi-user support for CASE over multiple methods. Transformation-based schemes (CDIF, federated CASE) cannot provide a true solution, although they may be useful in certain situations. Open CASE theory useful for multiple tools, methods and users, but lacking in ability to modify methods, and actual working implementations. MetaCASE useful for multiple methods and method modification, and has existing implementations, but needs multiple tools and multi-user support.

4 Research problem

Having stated our basic framework for research, and examined the current state of the field and its issues, we now collate together these issues to provide us with a research problem. To do this, we take two studies, each of which provides clear results and a good identification of problems, and agrees with the majority of other recent surveys and research in their area. The first examines the field of CASE (Sto93), the second (Gol93) the field of metaCASE. First, we shall briefly describe the research environment, as it naturally affects the choice of research problem and research methodology.

4.1 Research environment and limitations

This research has taken place at the University of Jyväskylä in the MetaPHOR research group, funded as the MetaPHOR and CAMSO projects (Lyy94, Lyy97). The aim in the MetaPHOR group has been to examine metaCASE and CAME and produce a working system to support them. The research builds on the group's work in the earlier SYTI project, which developed a prototype of the first graphical metaCASE environment, MetaEdit (Smo91a), which provides the same graphical support for metamodelling as for standard CASE modelling. Constructive research thus plays a central role in the research methodology of the MetaPHOR project, as part of a wider multi-methodological approach.

Because MetaEdit is the most familiar metaCASE tool for us and also among the best existing tools (admittedly according to our criteria!), it forms a natural starting point for continuing research in the MetaPHOR group. However, the decision was made to build upon the theories and experience developed earlier, rather than the implementation: in a sense, an application of the 'build one to throw away' principle. This would allow a freer hand in the design, in particular in the ability to include new ideas from others' research as well as our own. Thus while we use MetaEdit below as a measuring stick by which to judge the performance of metaCASE tools according to the criteria developed by empirical researchers, this thesis's task is not to improve MetaEdit itself, but rather to improve *on* MetaEdit, and other metaCASE tools.

Whilst metaCASE may be considered the core area of research of the MetaPHOR research group, the research and integration of results on other related areas is of equal importance. Such related areas include process and agent modelling (Mar94), the method engineering process (Tol94), tools and metrics for method engineering (Ros95), and hypertext and design rationale (Oin97). The general omission of these topics from this thesis does not therefore imply a perceived lack of importance; they are simply being studied by the other researchers referenced above. Thus I refer to these only as necessary, and do not intend to make significant research contributions directly in these areas, although advances in the underlying repository or meta-model will of course have beneficial effects in some of them, particularly the method engineering tools and process.

It was already decided at an early stage that this thesis would be in the form of a collection of articles, rather than a monograph. This has the benefit that the articles could be published over the course of the research, making the research results available to other researchers, and also obtaining feedback from them. The weaknesses of such a thesis format are the inevitable repetition of some basic points in the introductions of each article, and the possibility of some information in earlier articles becoming outdated by the time the whole thesis is ready. In a fast-moving field such as metaCASE, especially where lack of communication between researchers has been a problem, it is to be hoped that these benefits to my research and the research community outweigh the weaknesses in the format.

These facts should be borne in mind as we develop our research questions below.

4.2 Input from CASE

Of the many sources of data on CASE use, Stobart et al. (Sto93) provide us with a good, recent, multi-national survey of CASE tool use. Much other survey work has been limited to the North European countries, often the Netherlands, limiting the trust that can be placed in the generalisability of the results. Stobart et al. use four research groups in three countries: Australia, the UK, and the Netherlands. Their findings are hardly surprising, and correspond closely with those of other recent surveys. They summarise their overall, world-wide findings on CASE tool problems thus:

1. tools do not offer the possibility to use several methods in an integrated manner
2. the integration between tools for different software development phases is poor
3. tools often require a change to methods other than those used within the organisation
4. tools for project management are not integrated with tools for the other software development activities
5. the coupling of design data to the tool repository is weak
6. there are no multi-user application development possibilities for communication between the CASE tool users

The first and second points require horizontal and vertical **method integration**. The third point requires **flexible customisation of method support**. The second and fourth points requires better **integration of different tools**, from different levels and phases of ISD. The fifth point, together with the previous requirement, motivates the separation of the conceptual data in the repository from the representational data shown by the tools, so that all representational data has a corresponding conceptual form in the repository (from the fifth point), but there can be many representations of the same concept (to allow the

integration between data in different tools mentioned in the second and fourth points). The **data model** used in the repository must be made **more powerful** to enable the representation of all the kinds of information necessary. Together these points motivate the need for **different tools** to represent the different representational paradigms used by methods at different phases and levels of ISD. The sixth point requires a **multi-user** tool with conceptual information shared at a fine granularity of data (for maximum ability of users to work together simultaneously) and update intervals (to ensure work is on the most up-to-date, released information).

4.3 Input from metaCASE

Goldkuhl and Cronholm present a framework for the design and evaluation of metaCASE environments (Gol93). Their findings correspond with those of other researchers (e.g. Mar93, Lou92). They derive the following requirements for next generation metaCASE:

1. multiple simultaneous representational paradigms: 'graphical, textual, matrices, tables etc.' (Gol93)
2. better graphical symbol support
3. integration between methods
4. underlying conceptual metamodel stored in the CASE repository
5. flexible way of working for different analysis strategies
6. better rules and syntax checking
7. rich and powerful metametamodel
8. method definition approach easy to learn and use

It is interesting to note that by their third point they rule out CASE application generators that produce a new CASE tool to support a method, and instead favour integrated metaCASE and CASE environments. In examining several currently available metaCASE tools they find the textual metamodelling languages hard to use, but praise MetaEdit's graphical metamodelling as 'user-friendly and promising' (Gol93).

The current MetaEdit (Smo91a) does not satisfy points 1 (**multiple tools**), 3 (**method integration**), and 7 (**powerful data model**), whilst there could also be some improvements on the second and sixth points. There is no support at all for 1 and 3, and whilst OPRR is among the more powerful metametamodels, Smolander (Smo91b) recognises already at the time he defines it that it is not powerful enough to represent newer methods involving complex objects (e.g. Booch (Boo91)) or n-ary relationships (e.g. NIAM (Nij89)). Point 4 (**metamodel in repository**) is open to interpretation: certainly the metamodel is stored, but the use of a file-based system instead of a repository is a source of some major drawbacks in MetaEdit.

4.4 Research problem definition

Our research problem was stated as:

> *The research in this thesis aims towards metaCASE tools which would better answer the needs and criticisms of CASE tool users, in these two particular ways:*
>
> - *The process, concepts, and tools for metamodelling should be improved*
> - *The metaCASE tool should be capable of being more easily and accurately configured and tested for a wider range of interlinked, evolving methods*

From this, and the presented needs and criticisms of CASE and metaCASE tool users, we can now state more exactly the requirements for these better metaCASE tools. The new metaCASE tools must offer:

- a more powerful common underlying conceptual meta-metamodel, with complex objects and n-ary relationships, and supporting the following three requirements:
- multiple tools for different representation paradigms (diagram, matrix, table, (hyper)text, etc.);
- multiple methods, fully integrated and customisable incrementally side-by-side with models, following engineering principles;
- simultaneous multi-user access at fine granularity via a repository, allowing reuse and linking to other users' work.

5 Research methodology

Having identified our research problems and described the environment in which the research takes place, we can now move to selecting a research methodology that will best direct and describe the way we address the research problems.

5.1 Choice and description of methodology

Wynekoop and Conger (Wyn91) divide CASE research along two dimensions, research method and research purpose. They adapt Scott Morton's classification of research methods (Mor85) to give eight research methods: case study, field study, laboratory experiment, action research, survey and basic research, which together account for 20% of the research analysed, and applied research and normative writings, which account for 80%. They identify this 'bias' as 'stalling more rapid development' and 'limiting what can be learned for future development'. Their survey is based on research on CASE published 1987–1989, mostly 1987 and 1988. Of the 13 studies of CASE in use mentioned in Section 3, ten were published after this period, showing that the research world has responded to this imbalance and carried out investigations on how CASE is used, its successes and failures.

These studies of practice thus provide us with the data we need to motivate future development. Clearly the process of assessment cannot continue indefinitely with no new constructive research: at some point we need to apply these research findings to make a new generation of CASE tools, or in this case, metaCASE tools.

Whilst Wynekoop and Conger provide categories for constructive research, their framework is subtly changed from that of Scott Morton, and even throughout the course of their paper, with the result that the number of categories classifying constructive and related research is reduced while that of evaluative research grows. This renders their framework less useful than it could have been for classifying research of a constructive nature.

Nunamaker, Chen, and Purdin (Nun91, Nun92) provide a research framework particularly designed for research involving systems development or construction. They defend systems development as a research methodology, showing how it fits into the common research pattern of 'problem, hypothesis, analysis, argument' by likening it to 'proof-by-demonstration'. They stress the importance of a multi-methodological approach that integrates four strategies: **theory building**, **systems development**, **observation**, and **experimentation**. Research in the strategies is mutually supportive (Figure 4): a theory "suggests insights on issues, supports and is supported by tool development, and is both a precursor for and an outcome of experimentation and observation" (Nun92).

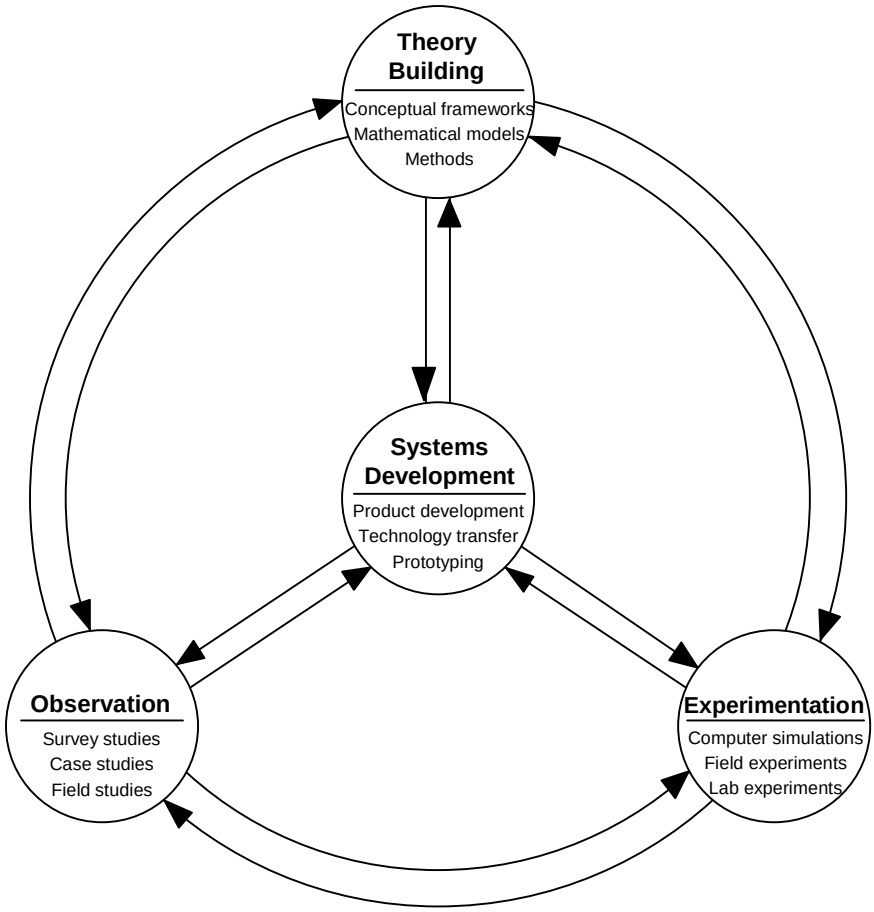


FIGURE 4 A multi-methodological approach to IS research (Nun91 p.94)

Theory building includes the development of new ideas and concepts, and construction of conceptual frameworks, new methods or models, e.g. data models. Theories can suggest research hypotheses, guide and enable research.

Experimentation includes research methods such as action research, laboratory, field and simulation experiments. It may attempt to validate theories, or look at issues of acceptance and technology transfer. Its results can be used to refine theories and improve systems.

Observation relies on unobtrusive research methods such as case studies, field studies and surveys. It is often used when little is known in a research area, to provide hypotheses for testing or to focus later research.

Systems development consists of five stages: **concept design**, **architecture construction**, **prototyping**, **product development**, and **technology transfer**. These largely correspond to the five phases of the overall systems development research methodology, where prototyping and product development are subsumed under product development and preceded by system analysis and design. This imperfect matching appears to have adversely affected the discussion of the phases, where some of the clarity and obvious application present in the figure and its four strategies is lost. The basic content of the phases remains clear.

1. **Construct a conceptual framework and decide a research problem**
   The conceptual framework serves as a foundation for the research in terms of direction and description. The examination of the field necessary to such a framework provides an understanding of the situation and is used to generate a research problem. The problem then motivates theory building within the conceptual framework.
2. **Develop a system architecture**
   The architecture provides a route map for the system building process, and shows how the various components relate and interact. The environment constraints, development objectives, and resulting system's functionality should be stated, with verifiable requirements. The emphasis will often be on reporting research on "new functionalities or innovative user interface features" (Nun91, p.99) rather than system speed.
3. **Analyze and design the system**
   This involves the understanding of the domain, the application of relevant scientific and technical knowledge, and the creation, synthesis and evaluation of alternatives. The design should be theory-based and modelled abstractly, rather than environment-specific. Data structures should be considered and specified.
4. **Build the system**
   A prototype can demonstrate feasibility and prove theories about functionality. For real-world testing the prototype, if it proves successful, should be developed into a product and taken into use in an organisation. The implementation process provides useful feedback to earlier phases.
5. **Experiment, observe and evaluate the system**
   Once there is a working system, it can be tested against the requirements, and its effectiveness in use examined by empirical studies. The results of the tests are fed back into further development of the system, the concepts, and the hypotheses.

Systems development research must conform to five criteria (Nun91, p.101):

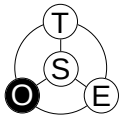
- The purpose is to study an important phenomenon in areas of information systems through system building
- The results make a significant contribution to the domain
- The system is testable against all the stated objectives and requirements
- The system provides better solutions to IS problems than existing systems
- Experience and design expertise gained from building the system can be generalised for future use.

The constructive or systems development approach is described as "a critical contributor among the methodologies available" (Nun91). As our research fulfils the five criteria above, and the methodology otherwise provides a good framework for this research, we adopt the systems development research

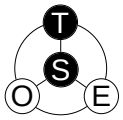
methodology as a framework for the description of the research in this thesis, and also to motivate and direct the research itself.

5.2 Application of the methodology in this research

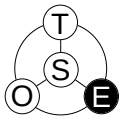
First we shall look at the research prior to this thesis, which provides its starting point. As can be seen below, this forms one whole cycle of our research methodology. Second, we examine the research presented in this thesis, which provides a second complete cycle through the methodology. The icons on the left show roughly how much each type of research strategy is concentrated on in the adjacent paragraph. **Bold** text in the paragraphs shows a research strategy, and *underlined italic* text refers to a research question.



Looking at the state of the practical and research field of CASE at the start, we saw that there are many CASE tools in use, but that they were not producing the needed benefits. The many case studies, field studies and surveys on CASE in use provided us with a large body of **observation** research, which motivates and directs our research.

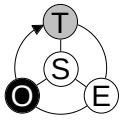


The conceptual framework for ISD used in the MetaPHOR project comes from the OPRR model (Smo91b), and its application in the MetaEdit **system's development** (Smo91a). The research on MetaEdit presented in (Smo91a) gives us a **theoretical** framework with three levels of ISD (metametamodel, metamodel and model levels) and two dimensions of information in a metaCASE tool (type-instance and conceptual-representational).

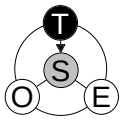


The commercial version of MetaEdit has been taken into use in over 30 countries, and used to model over 50 methods (graph types). Its use has also been examined in **experimental** situations (Gol93, Ros94a).

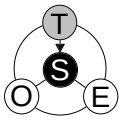
Thus, the starting point for this work is **observation** of existing CASE tool use and problems (e.g. Nor89, Aae91, Kus93, Sto93), some **theory** on metaCASE, a **prototype system**, and some **experimentation** with this system (Gol93, Ros94a) and other similar systems (Gol93, Mar93). This provides us with a first cycle of incremental systems development, including all four strategies, from which to launch this second cycle.



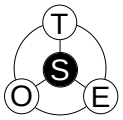
The introduction to this thesis looks at the existing situation and history of CASE and metaCASE, and outlines the current needs for development in the field (**observation**). Some preliminary **theories** and hypotheses are built, and the conceptual framework for the research established (the first phase in the methodology). The analysis of the needs motivates some ideas for the second phase of the methodology, architecture development, and *provides our three research questions*: a better data model, multiple tool support, and multiple, integrated method support.



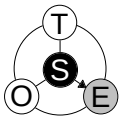
The first paper, What's in a Relationship, is motivated by the need for improvements in the *meta-metamodel* of metaCASE tools on a conceptual level, as stated in the first paper. These improvements give clarity and ease to metamodelling, and also provide beneficial results for implementation, in particular by allowing incremental method engineering, and improving the support for co-operative multi-user work. The paper thus builds **theories** to support the design of the system, and helps in the development of the conceptual framework (the first phase in the methodology). As it extends to considering data structures, it also covers part of the third phase of the methodology and some **systems development**.



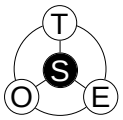
The second paper, MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment, summarises and explains the problems of existing CASE technology, and presents the GOPRR *meta-metamodel*, architecture and metamodelling tools of MetaEdit+. It shows how GOPRR and the metamodelling tools support the incremental creation and use of *multiple integrated methods*. The description of GOPRR extends the **theory** presented in the previous paper from relationships to the whole meta-metamodel, and the architecture and metamodelling tools represent the result of the **systems development** of MetaEdit+.



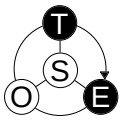
The third paper, MetaEdit+: CASE Functionality to Support Production, Coordination and Organizational Control and Innovation describes the MetaEngine, the functional heart of MetaEdit+, and its tool integration strategy. It further concisely describes each modelling tool in MetaEdit+, thus complementing the previous paper, which described the metamodelling tools. The main focus is thus on the *multiple tool support* of MetaEdit+, representing the **systems development** strategy, although there are elements of other research questions and strategies present.



The fourth paper, Application of Repository Technology and Concepts to a MetaCASE Environment, describes how MetaEdit+ supports *simultaneous multi-user* access for several modellers and meta-modellers. It takes a novel view of the environment by considering the database and MetaEngine as forming a repository analogous to standard generic commercial (business) repositories: MetaEdit+ can thus be viewed both as using and forming a repository. A particular emphasis is on the support for concurrency provided by a locking scheme abstracted away from tools into the MetaEngine, again aiding the integration of *multiple tools*. Whilst the main strategy is clearly **systems** development, the paper includes a measure of **experimentation** by evaluating MetaEdit+'s repository functionality according to Bernstein's criteria, and its support for collaborative work according to the criteria of Vessey and Sravanapudi. The development of a new high-concurrency updatable collection could further be regarded as theory building.



The fifth paper, A Matrix Editor for a MetaCASE Environment, goes on to show how one particular tool was added to the MetaEdit+ environment. It describes the requirements analysis, high-level and user-interface design of the new matrix manipulation tool. The paper shows how the matrix editor works as a tool, and how it is integrated into the *multiple tool support* of MetaEdit+: this also provides some verification of the suitability of GOPRR for multiple representation paradigms, and the success of the MetaEngine approach to adding new tools. In addition to the paper's main strategy of **systems development**, some preliminary hypotheses are presented on the usability of matrices for metamodelling as well as modelling.



The sixth and final paper, Evaluating Method Engineer Performance: An Error Classification and Preliminary Empirical Study, extends the need for *multiple tools* to metamodelling. It investigates the evaluation of method engineer's performance, extending Batra's **theory** of analysing database models and applying it to metamodelling to develop a classification to identify and quantify errors. The classification is used in an **experiment** that compares the performance of method engineers using a graphical diagram or matrix-based tool to design metamodels. It thus also addresses the need for better *support for method engineering* by showing that different tools are better for different parts of the metamodelling process.

From the icons by the side of each paper, showing the main and subsidiary research strategies used and the relations between them, it is clear that the research has followed a multi-methodological systems development approach as advocated in (Nun91), both in terms of the phases of the research and the use of the different strategies to support, motivate and build on each other.

All four research strategies have been used, with an emphasis in this thesis on systems development and theory building, reflecting my role in the wider MetaPHOR project. This reflects the progress of the research through the five phases of the framework: the multi-user version of MetaEdit+ was finished at the end of 1996, and thus observation is only just becoming possible. Similarly, to date there has been only one course on metamodelling at this university, providing a suitable research setting and subjects with which to study metamodelling experimentally. Experimentation and observation will thus form dominant strategies in research in the near future, as the current cycle of incremental systems development is concluded and the next begins.

6 Summary of the thesis

In this section we list the six papers that make up the body of the thesis, along with brief descriptions of the problems addressed and results of each. The publication details of the papers and authors are listed for each paper, and the division of work among co-authors for the group articles is described at the end of the section.

Note that the order is logical rather than chronological. In this respect, the positioning of the article on the Matrix Editor is somewhat difficult: the architecture into which the Matrix Editor fits is finalised and described in papers published later, and this article was written before the Matrix Editor was completed. However, it still seems more logical to position it after the description of MetaEdit+ as a whole, and before the empirical investigation of some claims made in it about the usability of the matrix format for metamodelling.

6.1 What's in a Relationship? On Distinguishing Property Holding and Object Binding

Proceedings, 3rd Conference on Information Systems Concepts (ISCO3)
Steven Kelly

Research problems and methodology

Current data models, or meta-metamodels, used for CASE and metaCASE are not powerful enough to describe and support the concepts needed for modern methods. In particular, many do not offer full support for n-ary relationships, decomposition of objects with proper handling of interface relationships, allowing reuse, polymorphism of metatypes, and modelling of methods such as Data Flow Diagrams (Gan79), where relationship bindings are not fully symmetrical. These deficiencies present a major barrier to the realisation of a metaCASE environment allowing modelling of multiple, integrated methods, multiple tools, and multiple users. Many of the problems in the data model are concerned with the nature of the concept or metatype of relationship, and thus the semantics and behaviour of relationships in different data models are examined, to identify the source of the deficiencies and develop a better theory and framework of relationships in CASE and metaCASE.

Research results

The paper identifies fundamental problems in the applicability of the currently used underlying data models for CASE and metaCASE. The concept of relationship is seen to have been overloaded by the inclusion of property holding in addition to its behaviour of binding objects together. The negative consequences of this are demonstrated: the underlying similarities of all

property-holders (objects, relationships and roles) are obscured; the role and relationship type definitions are too tightly bound to each other and object types, preventing their reuse; binding information and handling is distributed and partially duplicated in each of these metatypes, leading to inefficiency and dangers of inconsistency; in a multi-user environment, duplication of information increases the extent of locking requirements, reducing the bandwidth of simultaneous work; modelling methods such as Data Flow Diagrams (Gan79) requires the construction of dummy relationship types to make up for the allocation of only one pair of roles and objects to each relationship type.

A new model is presented which uses a separate construct, Binding, to handle the actual linking of objects, whilst maintaining the concepts of relationship and role to hold properties along the path of the binding. The existing and new models are compared with respect to modelling accuracy and comprehensibility, and to how they perform in multi-method, multi-tool, multi-user situations, and the new model shown to be a significant improvement. The place and effectiveness of bindings in the GOPRR data model is shown, and the possibilities for their use to solve complicated problems of metatype polymorphism in methods such as NIAM are demonstrated.

6.2 MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment

Proceedings of 8th International Conference on Advanced Information Systems Engineering, CAiSE'96
Steven Kelly, Kalle Lyytinen and Matti Rossi

Research problems and methodology

The paper addresses the problems of supporting software engineering in-the-large, by many, and over time. It examines the history and current state of CASE technology, and identifies major deficiencies in the support for integrating methods, alternative representation paradigms, and multiple users. A further serious problem is the rigid method and process built into the tool. MetaCASE tools have offered some solutions, but their metamodelling languages have been hard to use and lacked the power to adequately describe methods. The paper seeks to advance current theory with the development of the GOPRR metamodelling language, and describes the architecture of a CASE and metaCASE environment, MetaEdit+, and its CAME tools that allow metamodelling with GOPRR via an easy form-based GUI.

Research results

GOPRR extends the existing OPRR meta-metamodel, itself a kind of extended ER model, with a concept of Graph, mapping to the definition of e.g. Data Flow

Diagrams. This allows the representation of multiple methods and models while still maintaining each as an identifiable whole. In addition, GOPRR is object-oriented, providing the power of inheritance and polymorphism for the metamodeller. The ability to define links other than relationships between model components is identified as important, and several inter-graph link types are identified, including explosion, decomposition, component reuse and property sharing. The first two have been known for some time, but their exact definitions and the distinction between them are here made more precise: explosion is a simpler more navigational link between something in a graph and another graph, whereas decomposition represents the internal structure of an object, and also its interface relationships, allowing the decomposition graph to be reused in a similar manner to components in CAD or computer-aided chip design.

Component reuse is introduced as the ability to include a component from one graph in another graph, even another graph from a different method, providing that the type of the component is legal in the second method. This is made possible by the ability to reuse type-level components when defining the metamodel of the graph. Property sharing is the complementary ability to define that two individual components both have the same property in a certain of their property slots. This means that if the value is changed in one component, the change will also be visible in the other component. Property values can be constrained by rules, which are defined in a simple format similar to Backus-Naur. Finally, the paper proposes and shows a metamodelling paradigm that uses the GOPRR concept of Graph to model method diagrams and produce type level graphs, and allows the inclusion of graphs within other graphs as complex objects. This is demonstrated on the DFD method (Gan79).

The metamodelling tools of MetaEdit+ are presented, showing how their user-friendly form-based interface allows much faster metamodelling than has previously been possible. The support of GOPRR for evolution of metamodels with corresponding instantaneous updates of models is described: a major feature for method engineering, not present in any other environment.

6.3 MetaEdit+: CASE Functionality to Support Production, Coordination and Organizational Control and Innovation

Submitted to ACM Transactions on Software Engineering and Methodology
Steven Kelly, Kalle Lyytinen
Hui Liu, Pentti Marttiin, Harri Oinas-Kukkonen, Matti Rossi, Juha-Pekka Tolvanen

Research problems and methodology

Current CASE tools support a single fixed method at a time and a single representation paradigm (graphical). In practice, organisations often use a mix of existing methods, change methods as they take them into use, and different

users require different views on the same data at different times. This paper examines the needs for CASE functionality to support the production, co-ordination, organisational and learning/innovation functions which CASE should provide in organisations. It describes how MetaEdit+ supports these functions and features of organisational method use with a set of tools integrated by a common MetaEngine.

Research results

The paper describes the implemented MetaEdit+ system, in particular its architecture, MetaEngine, and various modelling tools. The architecture is based on principles of conceptual modeling, layered data base architectures, and object orientation, providing three kinds of independence that are vital to the success of the environment as a whole: data independence, representation independence, and level independence. Data independence separates the basic data required by the various tools from its storage and locking in the database; representation independence separates the conceptual data and its behaviour shared by all CASE editors from the representational (e.g. positional) data they need in their representational paradigm; level independence means that the environment follows the environment follows a symmetrical approach in its treatment of models and metamodels. Together, these abstract behaviour away from the tools into the MetaEngine, producing consistent behaviour across tools, and significantly reducing the work required to implement new tools. In addition, the MetaEngine automatically offers a large set of default behaviour that tools can obtain by simply subscribing to it. The MetaEngine is responsible for all operations on conceptual data, while each tool is only responsible for operations on its own representational data.

The various tools of MetaEdit+: Diagram, Matrix and Table Editors, Type and Graph Browsers, Hypertext subsystem, Design Rationale tool and Query Editor are described, and it is shown how together these form an integrated environment to support the functions and needs described above.

6.4 Application of Repository Technology and Concepts to a MetaCASE Environment

To be submitted to ACM Transactions on Software Engineering and Methodology
Steven Kelly

Research problems and methodology

MetaCASE tools have been hampered by their use of file-based data storage rather than a true repository, leading to lack of integration between methods, and by their support for only single users. This paper examines the user requirements for multi-user CASE and metaCASE, and the solutions necessary

to provide such support. It assesses the MetaEdit+ metaCASE environment as a repository for CASE data, following Bernstein's requirements for generic repositories (Ber96). It describes the multi-user support developed in MetaEdit+, in particular the locking strategies used, and a new data structure that was developed. It then assesses MetaEdit+'s support for multiple users according to the criteria used by Vessey and Sravanapudi (Ves95) in their comparison of several CASE tools support for collaboration.

Research results

MetaCASE tools can be usefully viewed as a repository for CASE data, and MetaEdit+ considered thus fulfils almost all of the criteria put forward by Bernstein for an ideal repository. Particular strengths include its support for multiple tools, its flexibility in supporting multiple methods and changes in methods on the fly, and support for reuse. Weaknesses were in the support for versioning and configuration management; however, Bernstein views these as the main mechanism for allowing simultaneous multi-user access, and this need is fulfilled in MetaEdit+ by a different mechanism: locking.

The locking system in MetaEdit+ is abstracted out of the tools into the MetaEngine, providing the locking behaviour for conceptual information and a generic set of locking functions for representational information. Simply marking menu items with the level of locking they require suffices to invoke those generic functions: if the locks are not present, the item is disabled. The locking strategy uses only write locks: no read locks are used, as only mild semantic inconsistency occurs if changes are made on the basis of old read information. This follows the same pattern of work that developers are accustomed to from before multi-user CASE: you work on the basis of the last released version. This use of only write locks coupled with the fine granularity of data (down to individual properties) allows a high degree of concurrency, solving the problems often found when trying to use standard database transactions for CASE.

A special set of locking behaviour is provided for metamodels, allowing the system administrator to choose whether metamodelling should only be allowed when no other users are logged in, whether only one metamodeller should be allowed but many simultaneous modellers, or whether there may be many simultaneous metamodellers and modellers. MetaEdit+ is thus the first tool to support metamodelling simultaneously with modelling, and multiple simultaneous metamodellers.

We describe a new collection data type, which was created to meet the needs of projects, i.e. sets of graphs. When a project is created, its set of graphs is small, and growing fast as users add many new graphs. As it gets older, the growth slows, and the normal maximum size is less than 100. The traditional solution to collections accessed by multiple users, B-trees, is highly inefficient at small sizes, allowing only a very low level of concurrent updates. The new collection type allows a high level of concurrent updates also when small, and behaves externally to tools that use it as any other normal collection.

Finally, MetaEdit+ is briefly assessed in its support for collaborative CASE. The results indicate an overall score higher than other tools assessed, even though MetaEdit+ supports metaCASE, which makes multi-user functionality more difficult.

6.5 A Matrix Editor for a MetaCASE Environment

Journal of Information and Software Technology, **36** (6) 1994
Steven Kelly

Research problems and methodology

Research in metaCASE or CASE shells has largely focused on supporting methods by allowing the definition of the concepts and representational symbols used in their diagrams. Little interest has been shown in environments supporting representational paradigms other than diagrams, such as matrices or (hyper-)text. The matrix in particular is often a better format for business information systems, metamodelling, and automatic algorithms for decomposing a system. This paper studies the requirements for such an editor, looking at methods that use matrices, previous use of matrices in CASE, and the metaCASE environment into which the matrix editor will be incorporated. These requirements are then used to motivate a high-level and user-interface design for the matrix editor, and this and the principles are demonstrated through several small examples.

Research results

The uses of matrices in CASE and metaCASE are analysed into three categories: methods that use matrices as their main representation paradigm, those that use mainly diagrams, but for which an alternative matrix representation exists, and those for which a matrix representation has not yet been used, but which could benefit from this alternative view. In the former two categories are both older structured methods and newer methods, especially business process engineering methods, where the matrix format helps in the division of a large number of processes and units into cohesive groups. In the last category, the matrix format is shown to be useful for metamodelling, particularly for showing the often complex rules concerning the legal bindings of relationship and object types.

The matrix is seen to be a useful format as it allows the display of a large amount of information which can be quickly and easily manipulated. In particular, it provides a more relationship-centred view of the data in a design graph, contrasting with the object-centred view of graphical diagrams. For example, adding a relationship in a matrix requires the selection of only one point, the intersection of the row and column of the objects in the relationship. The ease of automatic generation and layout of matrices means little extra user

work is needed to benefit from them: automatic layout of a diagram from a matrix is more complicated. Different automatic algorithms for grouping and arranging axis items can be provided, e.g. diagonalisation, affinity analysis and transitive closure.

The user interface described builds on existing similar paradigms such as spreadsheets, extending it with the possibility of hierarchical axes. The display of the elements in the matrix reflects their type definitions in the wider metaCASE environment, showing identifying properties, the graphical symbol defined for that type, single characters (e.g. CRUD matrices) or just an X. As the matrix editor is based on the GOPRR metametamodel, rather than any particular method, its user interface is flexible, automatically configuring itself to the method in use, whilst allowing users to influence the displayed information.

6.6 Evaluating Method Engineer Performance: An Error Classification and Preliminary Empirical Study

First version published in Proceedings of 2nd CAiSE / IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design.
Submitted as an invited paper (as one of the five best papers from the workshop) to The Electronic Journal of Information Systems Evaluation.
Steven Kelly and Matti Rossi

Research problems and methodology

Various ways of representing metamodels have been put forward, including textual, graphical and matrix. These different paradigms may have their own effects on how easily and well users can model methods. However, no comparisons of their performance have been performed, and indeed no way of comparing performance of metamodellers has been described. We extend Batra's classification of errors in data modelling to cover metamodelling, and use it to measure the performance of a group of metamodellers using either diagrams or matrices.

Research results

Batra's classification of different facets of database modelling forms a good basis, and we are able to modify and extend it to apply well to metamodelling. Our hypothesis was that the matrix users would perform better on modelling simple relationships, as the matrix format appears clearer and more compact, but in other facets of metamodelling diagrams would be easier to use, as users were generally already more familiar with the diagram format.

From a pilot group of ten users, we obtained results that appear to support the first half of the hypothesis, that matrices were better for simple

relationships, but the second half appeared unjustified: in the other facets overall there was only an insignificant benefit from using diagrams, with little differences excepting one facet whose sharp difference was ruled out because modelling of that facet was identical in both paradigms.

Thus the matrix format would appear to be useful for metamodelling, and there is probably a need for multiple formats, so that users can switch formats to use the best format for each facet. MetaEdit+ supports such switching of formats whilst modelling. The tentative results from this pilot study confirm the usefulness of the classification and the applicability of this experiment set-up.

6.7 About the joint articles

My contribution in the second paper, MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment, includes the whole of Sections 3 (architecture) and 4 (GOPRR). In addition, I was responsible for some of the functionality described in Section 5 (metamodelling tools), and for bringing together the paper as a whole.

In the third paper, MetaEdit+: CASE Functionality to Support Production, Coordination and Organizational Control and Innovation, I was responsible for Sections 3 and 4 (architecture and MetaEngine), and for bringing together the paper as a whole. I was also responsible for the Matrix Editor and for the more advanced stages of development of the Diagram Editor. Whilst the tools themselves were implemented by the various authors, and each implementer initially submitted some text on their tool, Sections 5 to 7 are almost entirely my own text, bringing together the descriptions of the tools and their functions in the wider environment. The introductory sections and conclusions were formed largely from Kalle Lyytinen's ideas and original text.

In the sixth paper, Evaluating Method Engineer Performance: An Error Classification and Preliminary Empirical Study, the division of work was largely even throughout. We collectively built the framework of the error classification, but the analysis and interpretation of the results was almost entirely my work.

7 Conclusion

7.1 Contribution of the thesis

The main contribution of this thesis lies in the development of metaCASE principles and solutions to better address the problems of CASE, and move towards CAME. In particular, we identify the current problems of CASE and metaCASE and their causes, and propose four main areas addressed in this thesis for developments of metaCASE: meta-metamodel extensions and improvements, multiple representational paradigm support, engineering and

integration of multiple methods, and multi-user support. Each of these areas is addressed in one or more papers, providing a deeper examination of the problem, directions for an overall solution, components of a specific solution, and examples of their applicability.

- The concept of relationship is analysed and honed for use in metaCASE by separating the property-holding and object-binding aspects. This new approach extends the GOPRR data model making it better able to support more complicated multiple integrated methods and multiple users.
- The concepts necessary for linking between methods are examined and refined, and a graph-based paradigm designed for incremental method engineering and metamodelling of multiple, integrated methods.
- GOPRR is made object-oriented, giving the power of inheritance, instantiation, abstraction, polymorphism and reuse to users to enhance the quality and speed of method engineering and software engineering.
- These extensions in GOPRR provide significant advances in the breadth and depth of coverage of methods which can be metamodelled without having to resort to programming or logic languages.
- MetaEdit+ is the first metaCASE tool where metamodelling and modelling take place within the same environment, with instant automatic updates of models to appropriately reflect metamodel changes.
- An object-oriented strategy is presented for collecting the central functionality of a metaCASE environment into a MetaEngine, including the GOPRR concepts and their behaviour, and other generic behaviour, dialogs etc. used in all tools. This provides automatic locking, type palette toolbar, hypertext functionality, and reuse functionality seamlessly in all editors, with virtually no coding in the editors. It also significantly improves consistency between the look and behaviour of the tools. The implementation of this central functionality in the MetaEdit+ MetaEngine allows quick creation and smooth integration of new editors, including the Matrix Editor described below.
- The central functionality of the MetaEdit+ environment is shown to be analogous to a repository and to perform well when evaluated as a repository. The multi-user support built appears to exceed that of other existing CASE tools examined. MetaEdit+ is the first environment to allow simultaneous metamodelling and modelling, and the first to allow multiple simultaneous metamodellers.
- A set of automatic locking strategies are defined that enable MetaEdit+ to offer a high level of concurrency whilst guaranteeing consistency. A new collection data structure is developed that allows high concurrency of updates even when small, improving on B-tree performance.
- A matrix editor is designed and integrated to function in parallel with existing diagram and other editors in a multi-tool metaCASE environment,

providing representational independence — the ability to view the same conceptual data in several representational paradigms.

- A classification of metamodelling errors is developed, along with a quantification which allows empirical evaluation of different metamodelling tools, users etc.
- The applicability of the matrix format for metamodelling as well as modelling is demonstrated by an experiment, which indicates the need to use different representational paradigms and tools to develop different aspects of the same metamodel.

All of these solutions have been implemented in the construction of the MetaEdit+ metaCASE environment, following the research methodology of systems development. This environment has been made available commercially via MetaCase Consulting since the end of 1996 (the single user version was released a year earlier), with the number of licenses sold well into three figures. MetaEdit+ thus represents a fully working and integrated product, not just a research prototype.

Perhaps the best way to summarise the contribution is via the research methodology. This thesis fulfils the five criteria for successful application of the methodology given in (Nun91, p.101, quoted here on p. 55). A lot has been learned during the development of MetaEdit+, and these theories, architectures and observations have been generalised and made available to other researchers via the published papers. The end result, MetaEdit+ itself, is available as a full product, which in the first place can have an immediate, concrete beneficial effect on the systems development process, and in the second place provide a valuable source both for testing and experimentation by researchers, and for observations on metaCASE in use in the real world. Such observations have been largely impossible before, because of the lack of usable tools. MetaEdit+ provides a solution by filling the gap between the unfinished research prototypes that are too unstable and only support part of the necessary functionality, and the commercial tools that are too hard to use, requiring textual programming to metamodel.

7.2 Directions for further research

MetaEdit+ provides an environment in which much future experimentation and observational research can be carried out. In particular, experiments should be carried out on the applicability of different paradigms of metamodelling: text-language based, graphical, matrix-based, form-based and example-based. These should be examined with respect to their readability, conciseness, and accuracy, and the process of metamodel creation in the different paradigms should be observed.

Similarly, the use of the matrix editor for different modelling tasks should be observed and compared with that of other representational paradigms such as diagrams and tables. The ways in which multiple users utilise data in a large project using integrated methods is also an area of interest.

Taking a broader approach, it would be instructive to perform a case study of the use of MetaEdit+ in a large organisation. This would provide a valuable source of information on metaCASE benefits on both the metamodelling and modelling levels. One leading telecommunications organisation, which uses MetaEdit+ for a project spread over several sites in Finland, Denmark, Germany and the United States, forms a potential candidate, and initial approaches have already been made.

These research strategies of experimentation and observation will form the conclusion of the current cycle of systems development, and prepare the way for the next. In terms of systems development, there are two main extensions to the current MetaEdit+ that are significant both in research terms and also from the feedback from existing users.

Firstly, an editor should be added for the time-based 'fence' diagrams of e.g. OMT and the Unified Modelling Language. These diagrams cannot properly be drawn with the existing Diagram Editor, and the concepts of time and spacial order present interesting questions about the division of information between conceptual and representational data. Conceptually the diagrams seem to provide no problems for GOPRR, but representations would rely on properties in a new way: to provide spatial information for objects rather than just text elements.

Secondly, a facility for importing and exporting models should be added: currently MetaEdit+ only supports this for metamodels. Whilst it is usually preferable to have all users accessing the same repository, rather than manually importing and exporting, there are situations where import and export are needed. This would require solution of several notable problems, not least the question of how much to export: how far should object reuse and property sharing links be followed? The problem is akin to that of versioning complex objects, to which no clear satisfactory general solution exists. The solution in MetaEdit+ would thus attempt to add an extension above the GOPRR meta-metamodel that would provide for version and configuration management, in addition to import and export. The tool support and functionality for these improvements could also probably be largely combined, following the MetaEngine approach used elsewhere in MetaEdit+.

A further piece of research would be an impartial in-depth comparison of the three leading metaCASE environments, their strengths and weaknesses in both metamodelling and modelling. The present author would however hope to be ineligible to carry out this particular piece of research, on grounds of partiality to one of the environments chosen.

References

Aae91 Aaen, Ivan, Carsten Sørensen, "*A CASE of Great Expectations,*" Scandinavian Journal of Information Systems 3(**1**) (1991) pp.3–23.

Aae92a Aaen, Ivan, Aila Siltanen, Carsten Sørensen and Veli-Pekka Tahvanainen, "A Tale of Two Countries: CASE Experiences and Expectations," in *The Impact of Computer Supported Technologies on Information Systems Development*, K. E. Kendall, K. Lyytinen and J. I. DeGross (Ed.), North-Holland, Amsterdam (1992).

Aae92b Aaen, Ivan, "CASE Tool Bootstrapping — how little strokes fell great oaks," in *Next Generation CASE Tools*, K. Lyytinen and V.-P. Tahvanainen (Ed.), IOS Press, Amsterdam, Netherlands (1992).

Aal93 Aalto, J.-M., "Experiences on Applying OMT to Large Scale Systems," in *Proceedings of the Seminar on Conceptual Modelling and Object-Oriented Programming*, A. Lehtola and J. Jokiniemi (Ed.), Finnish Artificial Intelligence Society (1993).

Ald91 Alderson, Albert, "Meta-CASE Technology," pp. 81-91 in *Software Development Environments and CASE Technology, Proceedings of European Symposium, Königswinter, June 17-19*, A. Endres and H. Weber (Ed.) No. 509, Springer-Verlag, Berlin (1991).

Alf77 Alford, M., "*A Requirements Engineering Methodology for Real Time Processing Requirements,*" IEEE Transactions on Software Engineering 3(**1**) (1977) pp.60–69.

Amu87 Amundsen, B., B. Christoffersen, "*Can Today's Design Tools Support an Integrated Design Method?,*" Norwegian Institute of Technology, Trondheim, Norway (1987).

Aur88 Auramäki, E., M. Leppänen and V. Savolainen, "*Universal Framework for Information Activities,*" DATA BASE (Winter 1988) pp.11–20.

Avi88 Avison, D. E., G. Fitzgerald, "*Information systems development current themes and future directions,*" Information and Software Technology 30(**8**) (1988) pp.458–466.

Ban91 Banker, R. D., R. J. Kauffman, "*Reuse and productivity in integrated Computer-Aided Software Engineering: An empirical study,*" MIS Quarterly 15(**3**) (1991) pp.375–402.

Ber89 Bergsten, Per, Janis Bubenko jr., Roland Dahl, Mats Gustafsson and Lars-Åke Johansson, "*RAMATIC — A CASE Shell for Implementation of Specific CASE Tools,*" SISU, Gothenburg (1989).

Ber96 Bernstein, P. A., "*The Repository: A Modern Vision,*" Database Programming & Design 9(**12**) (1996) pp.28–35.

Boo91 Booch, G., "*Object-Oriented Design With Applications,*" Benjamin / Cummings, Redwood City CA (1991).

Boo97 Booch, G., J. Rumbaugh and I. Jacobson, "*Unified Modeling Language v1.0,*" Proposal to OMG, Rational Software, available at http://www.rational.com, Santa Clara, US (1997).

Bos94 Bosua, Rachelle, Sjaak Brinkkemper, "CASE Tool Integration: A State of the Art Review," in *Proceedings of the 5th Workshop on the Next Generation of CASE Tools*, B. Theodoulidis (Ed.), Universiteit Twente, Enschede, the Netherlands (1994).

Bri89 Brinkkemper, S., M. de Lange, R. Looman and F. H. G. C. van der Steen, "On the Derivation of Method Companionship by Meta-Modelling," in *Third International Workshop on Computer-Aided Software Engineering, CASE'89*, J. Jenkins (Ed.), Imperial College, London, UK (1989).

Bri90 Brinkkemper, Sjaak, "*Formalisation of Information Systems Modelling*," Thesis Publishers, Amsterdam (1990).

Bri93 Brinkkemper, S., "*Integrating diagrams in CASE tools through modelling transparency*," Information and Software Technology 35(**2**) (1993) pp.100–105.

Bri95 Brinkkemper, S., R. Engmann, J. Kreuk and M. van Loon, "*Tools for Information System Design*," Course material: Maestro II manual, DMG, Dept. of Computer Science, University of Twente (1995).

Bro82 Brooks, F., "*the Mythical Man Month: Essays on Software Engineering*," Addison-Wesley, Reading, Mass, USA (1982).

Bub71 Bubenko, J. A., B. Langefors and A. Sølvberg, "*Computer-Aided Information Systems Analysis and Design*," Studentlitteratur, Nordforsk, Lund (1971).

Bub88 Bubenko, J. A., "*A Method Engineering Approach to Information Systems Development*," the proceedings of the IFIP WG8.1 Working Conference on Information Systems Development Process (1988) pp.167–186.

Bub92 Bubenko, J. A., B. Wangler, "Research Directions in Conceptual Specification Development," in *Conceptual Modelling, Databases and CASE: An Integrated View of Information Systems Development*, P. Loucopoulos and R. Zicari (Ed.), New York (1992).

Cai75 Caine, S. H., E. K. Gordon, "PDL — A tool for software design," in *Proceedings of the National Computer Conference*, AFIPS Press (1975).

CDI91 CDIF, "*CASE Data Interchange Format Interim Standards vol. 1-3*," Electronic Industries Association Engineering Department (1991).

Cha86 Charette, R., "*Software Engineering Environments, Concepts and Technology*," McGraw-Hill, New York, USA (1986).

Cha96 Chau, P. Y. K., "*An empirical investigation on factors affecting the acceptance of CASE by systems developers*," Information & Management 30(**6**) (1996) pp.269–280.

Che76 Chen, P. P., "*The Entity-Relationship Model: Toward a Unified View of Data*," ACM Transactions on Database Systems 1(**1**) (1976) pp.9–36.

Che89 Chen, Minder, Jay F. Nunamaker Jr., "METAPLEX: An integrated environment for organization and information systems development," pp. 141–151 in *Proceedings of the Tenth International Conference on Information Systems, December 4–6, 1989, Boston, Massachusetts*, J. I. DeGross, J. C. Henderson, and B. R. Konsynski (Ed.), ACM Press (1989).

Chi88 Chikofsky, E. J., B. L. Rubinstein, "*CASE: reliability engineering for information systems*," IEEE Software (March 1988) pp.11–16.

Fou87a Arthur Andersen Consulting, "*Foundation-Method/1: Information Planning, Version 8.0*," Arthur Andersen, Chicago (1987).

Fou87b Arthur Andersen Consulting, "*Foundation-Method/1: Documentation, Version 8.0*," Arthur Andersen, Chicago (1987).

Exc87 Index Technology Corporation, "*Excelerator Reference Guide*," Index Technology Corporation, Cambridge, USA (1987).

Cro94 Cronholm, S., G. Goldkuhl, "Meanings and motives of method customisation in CASE environments — observations and categorizations from an empirical study," in *Proceeding of the fifth workshop on the next generation of CASE tools*, B. Theodoulidis (Ed.), University of Twente, Twente (1994).

Cyb92 Cybulski, Jacob L., Karl Reed, "*A Hypertext-Based Software Engineering Environment*," IEEE Software (March 1992) pp.62–68.

Dav90 Davenport, T. H., J. E. Short, "*The New Industrial Engineering: Information Technology and Business Process Redesign*," Sloan Management Review (Summer 1990) pp.11–26.

Ebe97 Ebert, J., R. Süttenbach and I. Uhe, "Meta-CASE in Practice: a Case for KOGGE," pp. 203–216 in *Proceedings of CAiSE '97, Barcelona, Catalonia, Spain, June 16–20*, A. Olivé and J. A. Pastor (Ed.) Vol. 1250, Springer, Berlin (1997).

Emm97 Emmerich, W., J. Arlow, J. Madec and M. Phoenix, "*Tool Construction for the British Airways SEE with the O2 ODBMS*," Theory and Practice of Object Systems 3(**3**) (1997) pp.(to appear).

Ern93 Ernst and Young, "*Automated Methods Environment*," NAVIGATOR system series, release 2.0, Ernst & Young (1993).

Gan79 Gane, C., T. Sarson, "*Structured Systems Analysis: Tools and Techniques*," Prentice Hall, Englewood Cliffs, NJ (1979).

Gol93 Goldkuhl, Göran, Stefan Cronholm, "*Customizable CASE Environments: A Framework for Design and Evaluation*," Linköping University, Sweden (1993).

Gol90 Goldstein, Reuven, "*Methodologies and CASE Tools — The Missing Link (a metaCASE model for designing and applying methodologies)*," CASE '90, Position paper (1990).

GOO95 GOODSTEP_Project, "*The GOODSTEP Project Final Report*," ESPRIT Project 6115, http://www.dbis.informatik.uni-frankfurt.de/REPORTS/GOODSTEP/goodstep.html, University of Frankfurt, Germany (1995).

Gru95 Grundy, J. C., J. R. Venable, "Providing Integrated Support for Multiple Development Notations," pp. 255-268 in *Proceedings of the 7th International Conference on Advanced Information Systems Engineering, CAISE'95*, J. Iivari, K. Lyytinen and M. Rossi (Ed.), Springer-Verlag (1995).

Gru96a Grundy, J. C., J. R. Venable, J. G. Hosking and W. B. Mugridge, "Supporting Collaborative Work in Integrated Information Systems Engineering Environments," in *Proceedings of the 7th Workshop on the Next Generation of CASE Tools (NGCT'96), Crete, May 20-24* (1996).

Gru96b Grundy, J. C., J. R. Venable, "Towards an Integrated Environment for Method Engineering," pp. 45–62 in *Method Engineering '96: IFIP WG 8.1/8.2 Working Conference on Principles of Method Construction and Tool Support, Atlanta, August 26-28*, S. Brinkkemper, K. Lyytinen and R. Welke (Ed.), Chapman-Hall, London (1996).

Gul92 Gulla, Jon Atle, Odd Ivar Lindland and Geir Willumsen, "PPP: An Integrated CASE Environment," in *Next Generation CASE Tools*, K. Lyytinen and V.-P. Tahvanainen (Ed.), IOS Press, Amsterdam, The Netherlands (1992).

Hag95 Haggerty, James J., "*Analysis/Design Tool,*" Spinoff Magazine, NASA Publication NP-217 20(**1**) (1995) p.95. Text available as http://tommy.jsc.nasa.gov/~woodfill/SPACEED/SEHHTML/pg105s95.html

Hah96 Hahn, E. von, "*Meta-modeling in ConceptBase — Demonstrated on FUSION,*" Studienarbeit, TU München (1996).

Hai92 Haine, Peter, "Second Generation CASE: Can it be justified?," in *CASE: Current Practice, Future Prospects*, Kathy Spurr and Paul Layzell (Ed.), Wiley, Chichester, UK (1992).

Han94 Hanna, Mary, "*Repositories built on Kindergarten lesson: sharing eases development,*" Software Magazine 14(**9**) (1994) p.37. Text available as http://www.it.rit.edu/~tdw/icsa710/refs/hanna.htm

Har93 Harmsen, F., S. Brinkkemper, "Computer Aided Method Engineering based on existing Meta-CASE technology," in *Proceedings of the Fourth Workshop on The Next Generation of CASE Tools*, Sjaak Brinkkemper, Frank Harmsen (Ed.), Univ. of Twente, Enschede, the Netherlands (1993).

Har94 Harmsen, Frank, Sjaak Brinkkemper and Han Oei, "Situational Method Engineering for Information System Project Approaches," in *Methods and Associated Tools for the Information Systems Life Cycle (A-55)*, A. A. Verrijn-Stuart and T. W. Olle (Ed.), Elsevier Science B.V. (North-Holland) (1994).

Har97 Harmsen, A. F., "*Situational Method Engineering,*" Doctoral dissertation, University of Twente, Moret Ernst & Young, Utrecht The Netherlands (1997).

Hey93a Heym, M., "*Methoden-Engineering Spezifikation und Integration von Entwicklungsmethoden für Informationssysteme,*" Ph.D. Dissertation, Hochschule St.Gallen, Switzerland (1993).

Hey93b Heym, M., H. Österle, "*Computer-aided methodology engineering,*" Information and Software Technology 35(**6/7**) (1993) pp.345–354.

Hid93 Hidding, Gezinus J., Gwendolyn M. Freund and Johan K. Joseph, "*Modeling Large Processes with Task Packages,*" Workshop on Modeling in the Large, AAAI Conference, Washington, D.C. (1993).

Hoc86 Hochstettler, William Henry, "*A Model for Supporting Multiple Software Engineering Methods in a Software Environment*," UMI Dissertation Information Service, Ann Arbor, Michigan (1986).

Hof96 Hofstede, A. H. M. ter, T. F. Verhoef, "*Meta-CASE: Is the game worth the candle?*," Information Systems Journal 6(**1**) (1996) pp.41–68.

IBM90 IBM, "*Repository Manager/MVS. General Information Version 1 Release 2*," Technical report GC26-4608-1 (1990?).

IEE83 IEEE, "*IEEE Standard Glossary of Software Engineering Terminology*," The Institute of Electrical and Electronics Engineering, Inc. (1983).

IEE95 IEEE, "*IEEE Standard for Developing Software Life Cycle Processes*," Ref. 1074.1-1995 (1995).

ISO89 ISO, "*Information processing systems: Information Resource Dictionary System (IRDS) Framework*," Draft International Standard ISO/IEC DIS 10027, International Organization for Standardization (1989).

ISO95 ISO, "*Information technology — Software life cycle processes*," ISO/IEC 12207 (1995).

Jar96 Jarzabek, Stan, Tok Wang Ling, "*Model-based support for business re-engineering*," Information & Software Technology 38(**5**) (1996) pp.355–374.

Kel94a Kelly, Steven, Veli-Pekka Tahvanainen, "Support for Incremental Method Engineering and MetaCASE," in *Proceedings of the 5th Workshop on the Next Generation of CASE Tools*, B. Theodoulidis (Ed.), Universiteit Twente, Enschede, the Netherlands (1994).

Kel94b Kelly, Steven, "*A Matrix Editor for a MetaCASE Environment*," Information and Software Technology 36(**6**) (1994) pp.361–371.

Kel95 Kelly, S., "What's in a Relationship: on distinguishing property holding and object binding," in *Proceedings of 3rd International Conference on Information Systems Concepts, ISCO 3*, W. Hesse and E. Falkenberg (Ed.), University of Marburg, Lahn, Germany (1995).

Keu97 Keuffel, W., "*A Trio of Object-Modeling CASE Tools*," DBMS Online 10(**5**) (May 1997), http://www.dbmsmag.com/9705d08.html (13.8.1997).

Kin94 Kinnunen, Kimmo, Mauri Leppänen, "O/A Matrix and a Technique for Methodology Engineering," in *Proceedings of the Fourth International Conference on Information Systems Development*, J. Zupansis and S. Wrycza (Ed.), Moderna Organizacija, Kranj, Slovenia (1994).

Kos97 Koskinen, M., P. Marttiin, "Process Support in MetaCASE: Implementing the Conceptual Basis for Enactable Process Models in MetaEdit+," pp. 110–122 in *Proceedings of Software Engineering Environments, SEE'97, Gottbus, Germany*, Jürgen Ebert and Claus Lewerentz (Ed.), IEEE Computer Society Press, Los Alamitos, CA (1997).

Kot84 Kottemann, J. E., B. R. Konsynski, "Dynamic Metasystems for Information Systems Development," in *Proceedings of the Fifth International Conference on Information Systems* (1984).

Kum92 Kumar, Kuldeep, Richard J. Welke, "Methodology Engineering: A Proposal for Situation Specific Methodology Construction," in *Challenges and Strategies for Research in Systems Development*, Kottermann W. W. and Senn J. A. (Ed.), John Wiley & Sons, Washington (1992).

Kus93 Kusters, R. J., G. M. Wijers, "On the Practical Use of CASE Tools: Results of a survey," in *Proceedings of the 6th International Workshop on Computer-Aided Software Engineering, CASE93*, Hing-Yan Lee, Thomas F. Reid and Stan Jarzabek (Ed.), IEEE Computer Society (1993).

Leh87 Lehman, M., W. Turski, "*Essential Properties of IPSEs,*" ACM SIGSOFT Software Engineering Notes 12(**1**) (1987) pp.52–56.

Lep94 Leppänen, Mauri, "Metamodelling: Concept, Benefits and Pitfalls," in *Proceedings of the Fourth International Conference on Information Systems Development*, J. Zupansis and S. Wrycza (Ed.), Moderna Organizacija, Kranj, Slovenia (1994).

LeQ88 LeQuesne, P. N., "*Individual and Organisational Factors and the Design of IPSEs,*" The Computer Journal 31(**5**) (1988) pp.391–397.

LeQ90 LeQuesne, P. N., "*Individual and Organisational Factors in the Design of Integrated Project Support Environments,*" Ph.D. Thesis, London Business School (1990).

Lin90 Lindgreen, P., "*A Framework of Information Systems Concepts,*" Interim report of the IFIP WG8.1 Task Group FRISCO (1990).

Lo95 Lo, Pius, "*Graphical Interface for CASE Environment Definitions in MetaView,*" Master's Thesis, University of Alberta, Canada (1995).

Lou92 Loucopoulos, P., B. Theodoulidis, "CASE — Methods and Support Tools," in *Conceptual Modelling, Databases and CASE: An Integrated View of Information Systems Development*, P. Loucopoulos and R. Zicari (Ed.), New York (1992).

Luc89 Luchner, Petra, Günter R. Koch and Franz Engelmann, "Glueing CASE Tools Together in a Heterogeneous CASE Environment," in *Proceedings of CASE 1989, Kista, Sweden*, A. Qwerin and J. Bubenko jr. (Ed.), SISU (1989).

Lyy87a Lyytinen, Kalle, "*Different Perspectives on Information Systems: Problems and Solutions,*" ACM Computing Surveys 19(**1**) (1987) pp.5–46.

Lyy87b Lyytinen, Kalle, "A Taxonomic Perspective of Information Systems Development: Theoretical Constructs and Recommendations," in *Critical Issues in Information Systems Research*, R. J. Boland Jr. and R. A. Hirschheim (Ed.), John Wiley & Sons Ltd. (1987).

Lyy94 Lyytinen, K., P. Kerola, J. Kaipala, S. Kelly, J. Lehto, H. Liu, P. Marttiin, H. Oinas-Kukkonen, J. Pirhonen, M. Rossi, K. Smolander, V.-P. Tahvanainen and J.-P. Tolvanen, "*MetaPHOR: Metamodelling, Principles, Hypertext, Objects and Repositories,*" Technical Report TR-7, Department of Computer Science and Information Systems, University of Jyväskylä, Finland (1994).

Lyy97 Lyytinen, K., J. Kaipala, S. Kelly, M. Koskinen, H. Liu, J. Luoma, P. Marttiin, R. Pohjonen, H. Oinas-Kukkonen, M. Rossi, M. Somppi, V.-P. Tahvanainen and J.-P. Tolvanen, "*CAMSO: Computer Aided Modelling Support for Organisations,*" Technical Report TR-18, Department of Computer Science and Information Systems, University of Jyväskylä, Finland (1997).

Mar91 Marmolin, H., Y. Sundblad and B. Pehrson, "An Analysis of Design and Collaboration in a Distributed Environment," pp. 147–162 in *Proceedings of ECSCW '91 2nd European Conference on CSCW* (1991).

Mar92 Marttiin, Pentti, Kalle Lyytinen, Matti Rossi, Veli-Pekka Tahvanainen and Juha-Pekka Tolvanen, "Modeling requirements for future CASE: issues and implementation considerations," in *Proceedings of The 13th International Conference on Information Systems, Dec. 13–16, 1992, Dallas, Texas* (1992).

Mar93 Marttiin, Pentti, Matti Rossi, Veli-Pekka Tahvanainen and Kalle Lyytinen, "*A Comparative review of CASE shells: A preliminary framework and research outcomes,*" Information & Management 25 (1993) pp.11-31.

Mar94 Marttiin, Pentti, "*Methodology engineering in CASE shells: design issues and current practice,*" Licentiate thesis, Technical Report TR-4, Dept. of Computer Science and Information Systems, University of Jyväskylä, Finland (1994).

Mar96 Marttiin, P., F. Harmsen and M. Rossi, "Evaluation of two CAME environments using a functional framework: findings on Maestro II/Decamerone and MetaEdit+," pp. 63–86 in *Method Engineering, Principles of method construction and support, Proceedings of the Method Engineering '96, Proceedings of IFIP 8.1/8.2 Working Conference on Method Engineering*, S. Brinkkemper, K. Lyytinen and R. Welke (Ed.), Chapman-Hall, London (1996).

Mar97 Marttiin, P., "Can Process-Centred Environments Provide the Customised Process Support Needed in Metacase? A Literature Review," pp. 165–180 in *Proceedings of the First International Workshop on the Many Facets of Process Engineering, 22-23.9.97, Gammarth, Tunisia*, G. Grosz (Ed.), Laboratoire PGL, Tunis (1997).

May39 Maynard, H. B., G. J. Stegemerten, "*Operation Analysis,*" McGraw-Hill, New York (1939).

McC89 McClure, C., "*CASE is Software Automation,*" Prentice Hall, Englewood Cliffs, NJ (1989).

Mer91 Merbeth, G., "Maestro II — the integrated CASE system from Softlab (in German: Maestro II — das integrierte CASE-System von Softlab)," in *CASE Systeme und Werkzeuge, 3e Auflage*, H. Balzert (Ed.), BI Wissenschaftsverlag (1991).

Mor85 Morton, M. Scott, "The State of the Art of Research," in *The Information Systems Research Challenge*, F. W. McFarlan (Ed.), Harvard Business School, Boston (1985).

Nij89 Nijssen, G. M., T. A. Halpin, "*Conceptual Schema and Relational Database Design: A fact oriented approach,*" Prentice-Hall, Englewood Cliffs, NJ (1989).

Nor89 Norman, R. J., J. F. Nunamaker Jr., "*CASE Productivity Perceptions of Software Engineering Professionals,*" Communications of the ACM 32(**9**) (1989) pp.1102–1108.

Nun91 Nunamaker, Jay F., Minder Chen and Titus D. M. Purdin, "*Systems Development in Information Systems Research,*" Journal of Management Information Systems 7(**3**) (1991) pp.89–106.

Nun92 Nunamaker, J. F., "*Build and Learn, Evaluate and Learn,*" Informatica — The Journal of Management Information Systems Development 1(**1**) (1992) pp.1–6.

Oei94 Oei, J. L. H., E. D. Falkenberg, "Harmonisation of information systems modelling and specification techniques," in *Methods and Associated Tools for the Information Systems Life Cycle*, A. A. Verrijn-Stuart and T. W. Olle (Ed.), Elsevier Science publishers (1994).

Oin97 Oinas-Kukkonen, H., "*Improving the Functionality of Software Design Environments By Using Hypertext,*" Ph.D. Thesis, A 296, Dept. of Information Processing Science, University of Oulu, Finland (1997).

Oll82 Olle, T. W., H. G. Sol and A. A. Verrijn-Stuart, "*Proceedings of the IFIP WG 8.1 Working Conference on Comparative Review of Information Systems Design Methodologies,*" North-Holland, Amsterdam (1982).

Oll86 Olle, T. W., H. G. Sol and A. A. Verrijn-Stuart, "*Proceedings of the IFIP WG 8.1 Working Conference on Comparative Review of Information Systems Design Methodologies: Improving the Practice,*" North-Holland, Amsterdam (1986).

Oll92 Olle, T. W., "A Comparative Review of the ISO IRDS, the IBM Repository and the ECMA PCTE as a Vehicle for CASE Tools," in *CASE: Current Practice, Future Prospects*, Kathy Spurr and Paul Layzell (Ed.), Wiley (1992).

Pap94 Papachristos, S., W. A. Gray, "Federated CASE Environment System: Towards the Realisation of Open CASE Environments," in *Proceedings of the 5th Workshop on the Next Generation of CASE Tools*, B. Theodoulidis (Ed.), Universiteit Twente, Enschede, the Netherlands (1994).

Par90 Parkinson, John, "Making CASE Work," in *CASE on Trial*, K. Spurr and P. Layzell (Ed.), John Wiley & Sons, Chichester (1990).

Pau93 Paulk, Mark C., Bill Curtis, Mary Beth Chrissis and Charles V. Weber, "*Capability Maturity Model, Version 1.1,*" IEEE SOFTWARE 10(**4**) (1993) pp.18–27.

Poc91 Pocock, John N., "VSF and its Relationship to Open Systems and Standard Repositories," in *Software Development Environments and CASE Technology*, A. Endres, H. Weber (Ed.), Springer-Verlag, Berlin (1991).

Rev95 Revault, N., H. A. Sahraoui, G. Blain and J.-F. Perrot, "A Metamodeling Technique: The MetaGen system," in *TOOLS Europe'95 Proceedings*, Prentice Hall (1995).

Ros92 Rossi, M., M. Gustafsson, K. Smolander, L.-Å. Johansson and K. Lyytinen, "Metamodeling editor as a front end tool for a case-shell," pp. 547–567 in *Advanced Information Systems Engineering*, P. Loucopoulos (Ed.), Springer Verlag, Berlin, Germany (1992).

Ros94a Rossi, M., J.-P. Tolvanen, "*Metamodeling approach to method comparison: A survey of a set of ISD methods*," Working Paper WP-34, Dept. of Computer Science and Information Systems, University of Jyväskylä, Finland (1994).

Ros94b Rossi, M., S. Brinkkemper, "Metrics in Method Engineering," in *Proceedings of CAiSE'95*, Iivari et al. (Eds), Lecture Notes in Computer Science 932, Springer-Verlag, Berlin (1995).

Ros95 Rossi, M., "*CAME Tools for MetaEdit*," Licenciate thesis, Technical Report TR-9, Dept. of Computer Science and Information Systems, University of Jyväskylä, Finland (1995).

Rum91 Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, "*Object-Oriented Modeling and Design*," Prentice–Hall, Englewood Cliffs, NJ, USA (1991).

Rup95 Rupnik-Miklic, E., J. Zupancic, "*Experiences and expectations with CASE technology — an example from Slovenia*," Information & Management 28(**6**) (1995) pp.377–391.

Rus94 Rust, Kees, personal communication on Westmount CASE tools (1994).

Sae94 Saeki, Motoshi, Kuo Wenyin, "Specifying Software Specification & Design Methods," in *CAiSE '94 Proceedings*, Gerard Wijers, Sjaak Brinkkemper and Tony Wasserman (Ed.), Springer-Verlag, Berlin (1994).

Sav90 Savolainen, V., J. Geels and J. Niemeier, "*SESAM, the HECTOR Methods and Tools Database*," Fraunhofer Institute for Industrial Engineering, Stuttgart (1990).

Sav93 Savolainen, Vesa, "*Analysis of Decision Criteria for ISD Tool Selection*," Systems Science 19(**2**) (1993).

Sil90 Siltanen, Aila, "The impact of CASE tools on IS management," in *CASE on Trial*, K. Spurr and P. Layzell (Ed.), John Wiley & Sons, Chichester (1990).

Slo93 Slooten, Kees van, Sjaak Brinkkemper, "A Method Engineering Approach to Information Systems Development," in *Procs. of the IFIP WG 8.1 Working Conference on the Information Systems Development Process*, N. Prakash, C. Rolland, B. Pernici (Ed.), North-Holland, Amsterdam (1993).

Smi88 Smith, J. B., S. F. Weiss, "*Hypertext*," ACM Special Issue on Hypertext 31(**7**) (1988) pp.816–819.

Smo91a Smolander, Kari, Kalle Lyytinen, Veli-Pekka Tahvanainen and Pentti Marttiin, "MetaEdit — A Flexible Graphical Environment for Methodology Modelling," in *Advanced Information Systems Engineering, Proceedings of the Third International Conference CAiSE'91, Trondheim, Norway, May 1991*, R. Andersen, J. A. Bubenko jr. and A. Solvberg (Ed.), Springer-Verlag, Berlin (1991).

Smo91b Smolander, Kari, "OPRR: A Model for Modelling Systems Development Methods," in *Next Generation CASE Tools*, K. Lyytinen and V.-P. Tahvanainen (Ed.), IOS Press, Amsterdam, the Netherlands (1991).

Sor88 Sorenson, Paul G., Jean-Paul Tremblay and Andrew J. McAllister, "*The Metaview System for Many Specification Environments,*" IEEE SOFTWARE (March 1988) pp.30–38.

Ste93 Stegwee, Robert A., Ria M. C. van Waes, "Flexible CASE tools for Information Systems Planning," in *Computer-Aided Software Engineering — Issues and Trends for the 1990s and Beyond*, T. Bergin (Ed.), Idea Group Publishing (1993).

Sto93 Stobart, S. C., A. J. van Reeken, G. C. Low, J. J. M. Trienekens, J. O. Jenkins, J. B. Thompson and D. R. Jeffery, "An Empirical Evaluation of the Use of CASE Tools," in *Proceedings of the 6th International Workshop on Computer-Aided Software Engineering, CASE93*, Hing-Yan Lee, Thomas F. Reid and Stan Jarzabek (Ed.), IEEE Computer Society (1993).

Tag90 Tagg, B. S., "*Implementing Tool Support for Box Structures,*" IBM Systems Journal 29(**1**) (1990).

Tah90 Tahvanainen, V.-P., K. Smolander, "*An Annotated CASE Bibliography,*" ACM SIGSOFT Software Engineering Notes 15(**1**) (1990) pp.79–92.

Tai97 Taivalsaari, A., S. Vaaraniemi, "TDE: Supporting Geographically Distributed Software Design with Shared, Collaborative Workspaces," pp. 389–408 in *Proceedings of CAiSE '97, Barcelona, Catalonia, Spain, June 16–20*, A. Olivé and J. A. Pastor (Ed.) Vol. 1250, Springer, Berlin (1997).

Tei77 Teichroew, Daniel, Ernest A. Hershey III, "*PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems,*" IEEE Transactions on Software Engineering 3(**1**) (1977) pp.41–48.

Tei80 Teichroew, Daniel, Petar Macasovic, III Ernest A. Hershey and Yuzo Yamamoto, "Application of the entity-relationship approach to information processing systems modeling," in *Entity-Relationship Approach to Systems Analysis and Design*, P. P. Chen (Ed.), North-Holland (1980).

Tho89 Thomas, Ian, "*PCTE Interfaces Supporting Tools in Software-Engineering Environments,*" IEEE SOFTWARE (Nov. 15, 1989) pp.15–23.

Tol93 Tolvanen, J.-P., K. Lyytinen, "*Flexible method adaptation in CASE environments — The metamodeling approach,*" Scandinavian Journal of Information Systems 5(**1**) (1993) pp.51–77.

Tol94 Tolvanen, J.-P., "*Methodology Engineering in CASE: Towards an Incremental Approach,*" Licentiate Thesis, TR-5, Dept. of Computer Science and Information Systems, University of Jyväskylä, Finland (1994).

Tol95 Tolvanen, J.-P., "Incremental Method Development for Business Modelling: An Action Research Case Study," pp. 79-98 in *Proceedings of the 6th Workshop on the Next Generation of CASE Tools, NGCT'95*, G. Grosz (Ed.), University of Paris 1, Paris (1995).

Tol96 Tolvanen, J.-P., M. Rossi and H. Liu, "Method Engineering: Current research directions and implications for future research," pp. 296–317 in *Method Engineering, Principles of method construction and support, Proceedings of the Method Engineering '96, Proceedings of IFIP 8.1/8.2 Working Conference on Method Engineering*, S. Brinkkemper, K. Lyytinen and R. Welke (Ed.), Chapman-Hall (1996).

Urw95 Urwiler, R., N. K. Ramarapu, R. B. Wilkes and M. N. Frolick, "*Computer-aided software engineering: The determinants of an effective implementation strategy*," Information & Management 29(**4**) (1995) pp.215–225.

Ves92 Vessey, I., S. L. Järvenpää and N. Tractinsky, "*Evaluation of Vendor Products: CASE Tools as Methdology Companions*," Communications of the ACM 35(**4**) (1992) pp.90–105.

Ves95 Vessey, I., A. P. Sravanapudi, "*CASE tools as collaborative support technologies*," CACM 38(**1**) (1995) pp.83–95.

Was86 Wasserman, A. I., P. A. Pircher, "*A Graphical, Extensible Integrated Environment for Software Development*," Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments (December 1986) pp.131–142.

Wel83 Welke, R. J., "*IS/DSS DBMS Support For Information Systems Development*," DATA BASE MANAGEMENT (1983) pp.195–250.

Wel92 Welke, R. J., "The CASE Repository: More than another database application," in *Challenges and Strategies for Research in Systems Development*, William W. Cotterman and James A. Senn (Eds.), Wiley, Chichester UK (1992).

Wij91 Wijers, G. M., "*Modelling Support in Information Systems Development*," PhD Thesis, Thesis Publishers, Amsterdam (1991).

Wyn91 Wynekoop, J. L., S. A. Conger, "*A review of computer aided software engineering research methods*," Information Systems Research, IFIP (1991), pp. 301–320.

You86 Yourdon, E., "*Whatever Happened to Structured Analysis?*," Datamation (June 1986) pp.133–138.