CASE Environments and MetaCASE Tools

Hosein Isazadeh
David Alex Lamb
{isazaho,dalamb}@qucis.queensu.ca

February 24, 1997
External Technical Report
ISSN-0836-0227-
1997-403

Department of Computing and Information Science
Queen's University
Kingston, Ontario K7L 3N6

Document prepared February 24, 1997

Abstract

MetaCASE is a generic approach to computer-aided software engineering. In recent years MetaCASE tools have been developed both commercially and in research centers. Their usage domain varies from an all-purpose CASE to a software engineering teaching tool. They are claimed to provide solutions to some of the key problems surrounding adoption and use of traditional CASE tools. However very little work has been done in the examination and analysis of MetaCASE tools.

This paper examines various opinions about software engineering environments and MetaCASE tools attempting to investigate the truth in many of their claims. In addition, reports our on-going research in developing a framework for studying MetaCASE tools. This framework is a comparative examination and categorization of the existing MetaCASE tools based on their architecture. To establish practicality we have included reviews and comparisons of five sample MetaCASE tools. Finally, we provide a summary of our results and discuss open problems in this area.

Contents

1 Introduction

Building large-scale software systems is a difficult task. A discipline is required to guide teams of software developers towards building correct systems that are on-time and within budget. Software Engineering provides such a discipline by devising methodologies to be used throughout the software process, but they are not easily enforcible. Automation of the methodologies, known as CASE, promised to make this discipline enforcible by easing the tasks of software engineers, but its success is questionable.

CASE tools are large, complex, very labour-intensive, and extremely costly to produce and adopt. They provide much less than they promised, and what they provide is not easily usable. It is no wonder that CASE tools are not as widespread as once expected. Examination of this problem reveals that supported methodologies play key roles. CASE tools support a fixed number of methodologies but software development organizations dynamically change their adopted methodologies. MetaCASE technology approaches the methodology automation from a dynamic perspective.

MetaCASE tools allow definition and construction of CASE tools that support arbitrary methodologies. A CASE tool customizer first specifies the desired methodology and customizes the corresponding CASE tool. Then software developers use that CASE tool to develop software systems. An advantage of this approach is that the same tool is used with different methodologies, which in turn, reduces the learning curve and consequently the cost. Any desired methodology can be automated or modified by the developing organization which provides a dynamic capability in todays dynamic and competitive world. From another perspective this technology can be used as a practical teaching tool considering the shortened length of development and learning times that suits academic course periods.

In this paper we examine the field of software engineering environments to understand CASE concepts, capabilities, and shortcomings. The first section is organized according to the historical time-line of this field to introduce the necessary concepts and terminology, focusing on MetaCASE technology. We establish the need for a review of the existing MetaCASE tools by examining the existing reviews. Next we introduce a framework for studying MetaCASE tools which outlines their typical architecture. Section 2 describes the components of this architecture and presents our view on how these tools should be categorized based on their underlying data representation model: Entity-Relationship(ER)-based, Object-Oriented(OO)-based, and Graph-based.

Sections 3, 4, and 5 contain the review of the representative tools in each category. We have selected a research-oriented tool (Metaview) and a commercial tool (Toolbuilder) from the first category of tools for review. Among the tools of the second category we review a commercial tool called MetaEdit. From the final category we review a prototype tool from IBM (4thought)

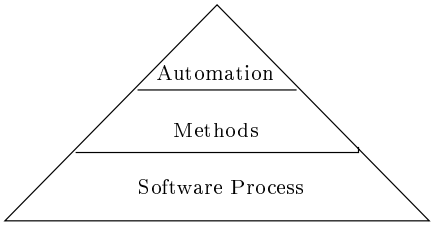

Figure 1: Charette's Model of Software Engineering Environment

and briefly examine a proposed tool from JRCASE in Australia called CASE-Maker. The reviews follow our framework by identifying and examining the typical components of the architecture. There are also tools not categorized at this point which are mentioned in section 6.

In section 7, we discuss and summarize the key features of the reviewed tools and identify their major shortcomings. The discussion also follows our component-based framework of study. This allows analysis and comparison of the components of each tool and identification of the open problems on a component basis. The final section of this paper provides some concluding remarks and the future directions of our research.

1.1 Software Engineering Environments (SEE)

Charette defines *software engineering environment*[1] to be the integration of "software process", "methods", and "automation" [11]. *Software process*[2] is the foundation of any environment and describes the sequence of events required to develop the software system. *Methods* are used to define, abstract, modify, refine, and document the software system. *Automation* is the computer implementation of the methods necessary to develop the software system. Figure 1 shows Charette's model of the software engineering environment. Ideally this model should be a rectangle where all the methods required by the software process are automated.

1.1.1 Software Process

Represented by process models or paradigms, software processes originate in the *monolithic waterfall model* [67]. This model regards development as one large process with successive phases as seen in figure 2. Each phase signifies activities that are distinct but the boundaries are fuzzy. Typically, a project begins with an *opportunity* or *feasibility study* which is the recognition of the problem and the feasibility of the solutions. Formulation of the user require-

[1]Sometimes called Software Development Environment (SDE).

[2]Also referred to as Software Lifecycle.

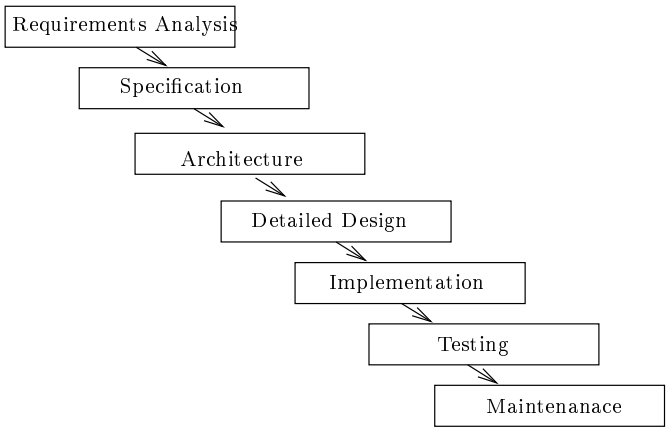

Figure 2: Waterfall Model of Software Process

ments and needs comes next and often produces a document called the *requirements specification*. This document specifies what is to be done as clearly as possible. Often formal and graphical notations are used to produce such documents [32]. *Design* phase starts with planning of all aspects of the project from the labour management and budget plans to software configuration management. The actual design includes an architectural design stage followed by a detailed design stage. The outcome is often a *design specification* document which specifies the modules in the system and their interfaces. *Implementation* and it testing deal with the development of the code and the verification and validation that the implemented code satisfies the specifications. *Maintenance* refers to the evolution of the system after the delivery which consumes a surprising 40 percent of total software effort in its lifetime. A detailed discussion of the activities involved in the software process can be found in [42].

With the waterfall model, the current position of the software system in the process is easily known. However activities such as verification and validation which cover the entire process are left outside this framework. Furthermore customer feedback is not possible until the completion of the development process.

Considering this, for large systems, an *incremental process* was used to allow quick prototyping and customer feedback every increment of the way [74]. Figure 3 illustrates this incremental approach in a very crude way. The introduction of high level 4th generation languages, such as TKL-TK, now allows the prototype to be used as the basis of the developed system, reducing the development time even further [59].

Today there are many software process models, among which the *knowledge-based* model is worth mentioning. A significant work in this area belongs to a ESPRIT[3] project called ASPIS [4]. In this approach, application domain

[3]European Strategic Program for Research and Development in Information

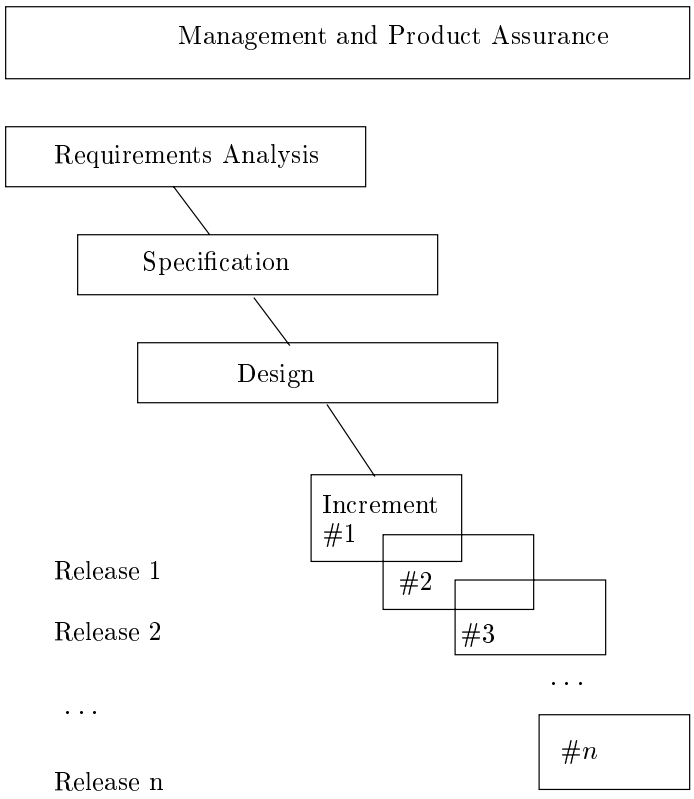

Figure 3: Incremental Model of Software Process

specific knowledge is captured and used to support the system development process. This knowledge is required in order to build a working quality system. Figure 4 shows a model of the knowledge-based approach as outlined by Anderson Consulting [18]. Different reasoning assistants (Requirements, Specification, and Implementation) communicate with the user, while using software information and domain specific knowledge. This approach may seem to be the only solution to building systems of enormous size. However as the size of the system increases, the amount of knowledge base also increases which reduces the performance. In general, current state-of-practice is far from a completely working, AI-supported, and automated knowledge-based process.

1.1.2 Methods

A *model* is a simple representation of the system. It provides insight about particular instances and collections of instances by abstracting away the nonessential details while generalizing the essential ones into the components of the model [60]. Methods are explicit steps and rules that are used to develop a

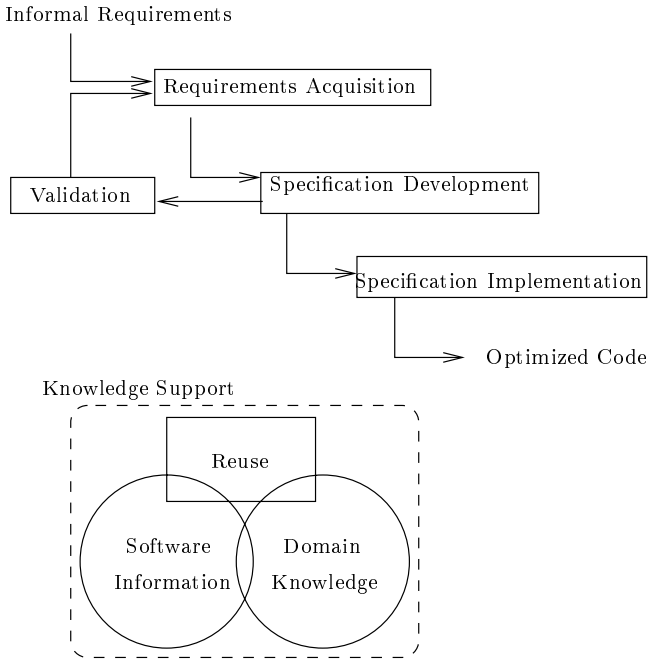

Figure 4: Simplified Knowledge Based Model of Software Process

model. They are required by the process model to provide reliability, efficiency, modifiability, or understandability when building a software system.

The ideal objective is to be able to use any method in conjunction with any other one. However methods have implicit and conflicting rules in them which may not allow their integration. Hence there have been a tremendous effort to choose, create, and combine integratable methods into methodologies that are useful throughout the software process. Therefore *methodologies* are defined to be "organized collections of methods" [11].

Examples of methodologies are: Jackson System Development (JSD) [35], Structured Systems Analysis and Design (SSAD) [90, 91], Booch Methodology [9], Jacobsen's Object-Oriented Software Engineering (OOSE or Use Cases Methodology) [36], Rumbaugh's Object Modeling Technique (OMT) [68], Sheller-Mellor Object-Oriented Analysis (OOA) [75], and numerous other ones. A problem, which is obvious from the number of the provided examples, is the great number and the huge variety of different types of methodologies. Each methodology has its own extensive set of specific notations, process rules, and guidelines. Therefore learning about various methodologies and switching between them is a very time-consuming and costly process. A recent ongoing effort by three of the leading methodologists (Booch, Jacobson and Rumbaugh) is the creation of a standard unified notation for object-oriented development [64]. However even if this notation could be used generically, non-object-oriented methodologies like JSD left out of this framework.

1.1.3 Automation

Having discussed software process and the supporting methods, we now focus on its practicality. Although methods that can improve the practice of software development have been available for nearly two decades, only in the last decade the daily practice of systems organizations has been changed [14, 39]. For many years the labour-intensive nature of such methods outweighed the improvements they produced. Automation has changed all that. In this context, automation differs from a single tool; it is the computer implementation of methods and an integral part of total process. It reduces the labour cost, increases productivity and creativity by putting the focus on the task, helps learning and communication, and allows use of certain methods that are clerically impossible for manual use.

In the following sections we will discuss the concept of automation in more detail with a focus on the "Computer Aided Software or Systems Engineering (CASE)" which is more along the theme of our research.

1.2 CASE Tools

One of the earliest and simplest attempts to automate certain aspects of the software process was the familiar UNIX *make* utility [27]. The early efforts were mostly focused on building a language specific programming support environment with facilities for error checking, debugging, compilation, linking, version control, and other supports [1, 82]. This is still an active research area of software engineering. What is changed however, is the focus which is moved from a particular phase of software process, namely implementation phase, to cover the entire process [23].

In the last decade various process frameworks and enhancements to software development models have gained acceptance, methods and methodologies are developed and practiced in software development projects, and well established techniques have been adopted from engineering disciplines. Many software engineering support tools have been used to aid the system development throughout the entire process from analysis and design phase to generation of code and testing. Since these tools allow the employment of well-known software engineering methods the term *Computer Aided Software Engineering (CASE)* has been coined [24].

"CASE is a production oriented integration technology that ties methods and tools into effective commercially viable environments" as defined by Chikofsky [14]. It offers graphical tools and manipulation capabilities to the developers of software systems. Typically CASE tools are based on some form of a "database", "dictionary" or "repository" that allows exchange of analysis and design objects. *Database, dictionary* and *repository* have different meanings in different contexts but for our purposes they refer to the storage

of analysis and design objects such as requirements statements, structured diagrams, and source code.

Most often system characteristics can be viewed from three perspectives. The first and most studied is the static structure of the system which refers to the data perspective. The second deals with the function of the system. The third perspective examines the dynamic behavior and looks at the control in the system [34]. CASE tools provide capabilities to represent these characteristics. In most cases Entity-Relationship (ER) diagrams and structured textual descriptions are used for the static structure [13]. The functions of the system are often represented by data flow diagrams [88]. The dynamic behavior of the system is usually captured by state transition diagrams [61].

1.2.1 The SMP Model of the CASE Tools

A recent definition of methodology divides it into three parts: "representation", "process", and "guidelines" [65]. The *representation* part is used to describe the components of the model with diagrams or textual notations. For example object-oriented methodologies provide diagrammatic and textual notations for defining the object structure of the model of the system. The *process* part provides the sequence of steps to be taken. An example would be the ordering of the definition of object structures which is done before defining the behavior. *Guidelines* refer to the set of heuristics, rules of thumb, and general directions that guide the developer in using the representation part and enforcing the process part. A good example of a guideline is the recognition of the objects from the user requirements specification. It is usually in a textual form with descriptions in natural language. Many object-oriented methodologies have guidelines that describe the extraction of frequent nouns from the requirements statement which become the object classes of the object structures.

On the other hand the *SMP model* [60] divides CASE tools into three interrelated components: "structures", "mechanisms", and "policies".

The first component relates to the *structures* such as the filesystems, abstract syntax trees or graph structures, project databases, and repositories that represent the basic software artifacts and other related information. The representation part of methodologies are usually supported by the structures of CASE tools. As an example, CASE tools supporting object-oriented methodologies define structures for classes, attributes, and relationships [73].

The second component deals with *mechanisms* such as the languages and tools that operate on the structures. They may be visible to the users or hidden as the low level support methods. Mechanisms encode information from all three parts of methodologies. As an example, most CASE tools provide languages that allow modeling of software systems according to the representation format prescribed by the methodologies. They may even enforce some

of the process rules and guidelines of the methodologies. We have observed, however, that the emphasis is on encoding the representation part more than the other two parts of methodologies.

The third component of CASE tools relates to the *policies* which are the rules, strategies, and guidelines imposed on the developers that may be supported or unsupported by the environment. The representation and process parts of methodologies often have rules that are supported by the policies of CASE tools. As an example, CASE tools may have rules that ensure the correctness of certain aspects of a diagramming representation notation or the ordering of the modeling process. In addition, guidelines prescribed by methodologies are also addressed by policies component of CASE tools but with much less emphasis. Some believe that none of the existing CASE tools support all the guidelines prescribed by their underlying methodology [73].

Based on our preliminary studies of guidelines suggested by the OMT methodology, we believe these guidelines range from simple general directions to difficult and tedious tasks. As an example, selecting meaningful names for classes is easily enforced by developers but may be difficult to ensure by CASE tools. On the other hand, selecting qualified names for objects can be a tedious task for developers but it is a simple job for CASE tools when they use a naming convention. Some of the guidelines are technically difficult to implement at this point. An example is extracting nouns from requirements specified in a natural language and mapping them onto classes of the modeled system. Understanding natural language is an active research area and we believe CASE research should also emphasis more on the application of this type of research in developing better CASE tools that support more of the guidelines.

1.2.2 Various Kinds of CASE Environment

Research in CASE tools has been a continuous focus of the scientific community. An early classification is based on the supported process phases [88]. *Upper CASE* or Front End tools are used during analysis and design; *Lower CASE* or the Back End tools are used during implementation and testing. This way of looking at CASE tools changed as soon as total process support tools were introduced. As an example of a typical CASE tool that supports the entire process, we can look at SoftDA [33]. As seen in figure 5, it has seven subsystems, two of which provide support for structured analysis and design [19]. Another two subsystems deal with detailed design and testing and the other three provide databases and reuse capabilities.

Further shift of research from ensuring that a CASE tool works was onto making sure that different tools work together. Hence the *integrated CASE tools* gained popularity to the point of having dedicated workshops and conferences. A landmark effort in this area is the categorization of tool integration

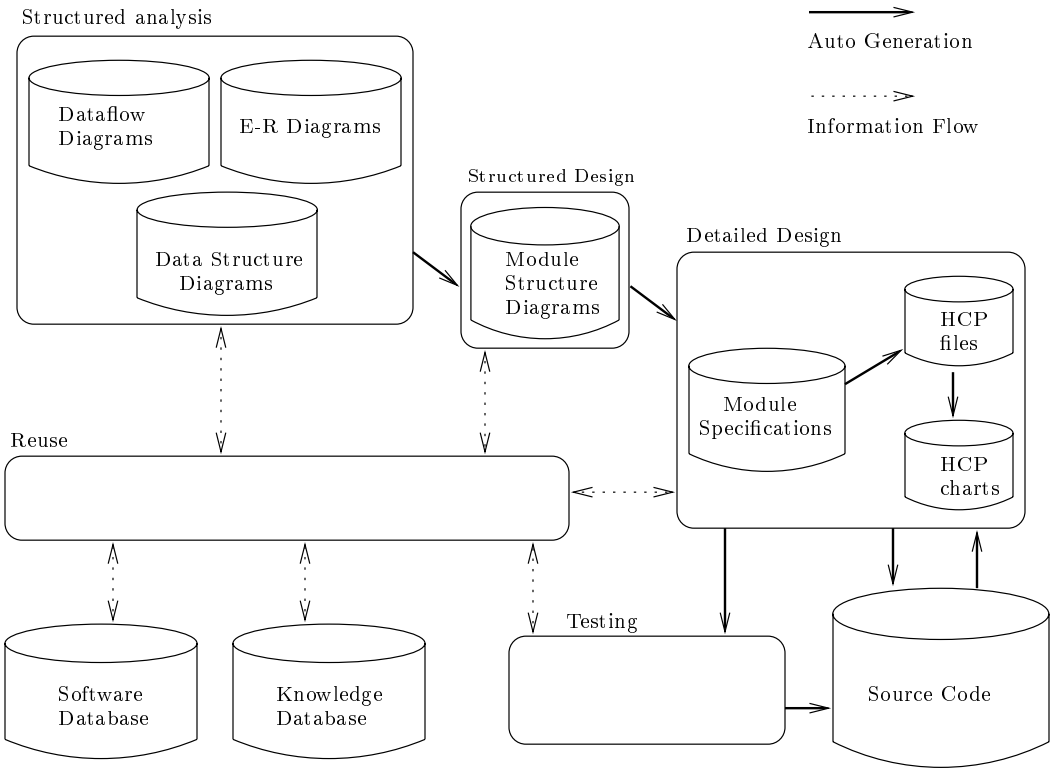

Figure 5: Simplified version of the SoftDA CASE tool functionalities

into five types by Wasserman: platform, presentation, process, data, and control integrations [87]. Tool integration means following open architectures principles, writing program interfaces, using file formats or database schemas, building data sharing mechanisms, and doing all of these within a common user interface.

Of other buzzwords involving CASE tools, three are worth mentioning. *Intelligent CASE tools* refer to the application of AI to CASE in the form of built-in domain specific (or real world) knowledge that may assist the developer [45]. These tools although promising and ambitious, lack performance when the domain is wide; they are not practical as general tools. *Repository-based CASE tools* offer enterprise-wide and project-wide repositories that integrate various tools for different stages of development [56]. However, they are very dependent on the employed methodology and can support development of only certain type of applications. "MetaCASE tools", also known as CASE shells, are the most recent approach to computer aided software engineering which we will discuss in the remaining of this paper.

1.3 MetaCASE Tools

Traditionally CASE tools only allowed the employment of a certain type of software engineering methodology which was fixed for the end users. With this approach the developer is provided with the expressive powers that the underlying methodology provides, a certain type of data dictionary or repository is used to store all the development artifacts, object Exchange is only allowed within the compatible and offered tools, and the offered graphical or textual editors are usable when the rules and guidelines of the methodology are followed. As an example, Teamwork/OOAD is heavily based on the Shlaer-Mellor methodology [30, 75].

This traditional approach is a source of great difficulty. It must be understood that every development company has its own organizational software process. CASE tool developers not only base their products on a particular methodology, they also adopt the underlying software process dictated by the methodology. This makes each CASE tool a special purpose tool that is useful for a particular type of organization, developing a particular type of software system. However, software developing organizations are different from one another and evolve over time. They change their product lines, management styles, and manufacturing procedures to adapt to their customer needs and to maintain a competitive edge. Therefore, since the software process and the development methodologies change, traditional CASE tools are not able to provide any realistic solutions.

Many CASE developers are now moving towards CASE tools that are capable of providing support for several different methodologies [26]. Often a group of related methodologies are supported by these tools without any real data or control integration. The user selects a methodology and follows the enforced notation, rules and guidelines. Although this approach is better than building tools that support a single methodology, it does not provide the solution to the problem of changing and dynamic software process which requires modifiable methodology support. We believe the solution is to build generic CASE tools that provide capabilities for dynamic production of different methodologies' toolsets.

Customizing a tool to an organization's needs is not a new concept. Many vendors have been providing this service to the large companies with a long software lifecycle who can afford the high cost of the customization. This high cost is due to the fact that CASE tools are large, complex, and very labour-intensive to produce. What is required is to provide the ability to capture the specifications of the required CASE tool and then to generate that CASE tool from its specification as automatically as possible. This is exactly what MetaCASE technology offers.

In general "a CASE Shell includes mechanisms to define a CASE tool for an arbitrary method or a chain of methods" as defined by Bubenko [10]. The

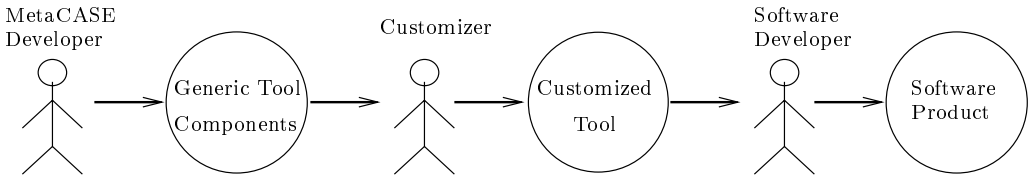

Figure 6: A High Level View of the MetaCASE Used in Three Levels

terms CASE Shell as used by Bubenko, MetaCASE tool as described earlier, and the metasystem as coined by those involved with the Metaview project [29], differ in detail but refer to the same concept of generic CASE tools. As a simple example of this technology, Alderson (who is associated with IPSYS, developers of Toolbuilder MetaCASE tool), describes compiler-compiler systems [2]. First the syntax of language is described to the system using a meta language. Then the system generates syntax and lexical analysis tables which parametrise a generic compiler to create a compiler for that specific language.

Alderson believes MetaCASE tools must have three components, a specification component, a generation component (which transforms the specification into parameters for the generic tool), and a run-time generic tool. A similar division is believed by Sorenson and Tremblay of Metaview system [78]. In their opinion there are three levels of specification in MetaCASE domain. In first level, the meta level, the meta definer defines the specification model or the "meta-model"[4] of the system. In second level, the environment level, the environment definer specifies the environment or the tool. In last level, the user level, the developer uses the specified tool to define a software system. Figure 6 shows a high level view of a typical MetaCASE tool with the three levels of usage.

At the top level MetaCASE developers build the generic components of the tool and define the basic structure of one or more meta-models to be used in capturing the representation information present in a methodology. In addition, mechanisms and languages are developed that would allow definitions of structures of the meta-models (according to the SMP model discussed in section 1.2.1). The difficulty in defining a meta-modeling technique is twofold. If the meta-model is too simple then it can be used to model most methodologies but it would not be sufficient when dealing with sophisticated methodologies. On the other hand, complicated meta-models are difficult to work with and might make the modeling of a new methodology too complicated and infeasible.

At the second level CASE customizers use the provided meta-model and the generic components to build a customized CASE tool. They define structures

[4] *Meta-model* is a term we will use to refer to the underlying data model of a methodology which is captured using data modeling techniques such as those based on entity-relationships.

of the particular CASE tool and encode the methodology prescribed representation and process rules (policies of SMP model). Customization of the CASE tool is done using the mechanisms and languages defined by MetaCASE developers. This process can take only a few hours which is much shorter and yet less costly than the traditional vendor customization approach [3].

At the last level software developers use the customized tool to develop software systems. An interesting approach to showing the powers of MetaCASE would be to compare development of the MetaCASE initially and using MetaCASE itself. This has been done in the case of Toolbuilder [28].

1.4 Existing Reviews

There exists a large amount of work on CASE tools including a number of books [52], various conference proceedings (SEE, SDE and CASE, and CAiSE), and special issues of journals and magazines. There are also many investigative and comparative papers [16, 57, 63, 85], and proposed general requirement models for CASE tools [23, 54]. However with respect to generic environments and MetaCASE technology there are only a few publications.

It is only recently in 1995 that a conference was dedicated to MetaCASE technology [55]. This focused effort brought together researchers from around the globe to exchange idea's and time will show its effects in the MetaCASE community. The keynote speaker, Alan Gillies believes MetaCASE will fail since the facilities offered by MetaCASE do not address the reasons why CASE tools are not adopted [31]. His reason is based on the 1991 Stobart et al's collected data about adopting CASE tools in UK [80]. He argues that only 5 percent of studied companies use CASE tools and the major reasons for this are said to relate to cost (30 percent) and lack of management backing (16 percent). Only 8 percent of the reason is due to the lack of supported methodologies which is the issue addressed by MetaCASE technology. However as we have discussed in section 1.3, high cost of adopting CASE tools is yet another issue that the MetaCASE technology addresses.

We believe MetaCASE enables companies to produce their customized CASE tools in a very cost efficient way. Unfortunately MetaCASE is a relatively a young field without much analytical data. However the MetaCASE conference proceedings alone contains over six papers dealing with adopting MetaCASE and using it to produce CASE tools. Often the selection criteria is the cost [3]. MetaCASE technology has also opened avenues of research in automation of such tedious jobs as the collection of measurement data, metrics, and assessments [22, 41]. This capability is due to the generic nature of the MetaCASE which allows methodology independent measurement data collection, generic metric engines, and quality assessments. In the MetaCASE conference many specific issues are addressed but there are no papers that provide a comparative review of the existing MetaCASE tools which is the

focus of our paper.

Among the existing publications about MetaCASE, there are two reviews that we will examine here.

1.4.1 Karrer and Scacchi's Review

This paper focuses on the broad area of the generic software engineering environments[5] and categorizes more than 60 of the related tools and technologies [38]. The basis for this classification is the provided service or the type of the adopted software process. This approach leads to division of meta-environments into five classes. Karrer and Scacchi discuss this division and name a few typical tools in each division. Since the number of the reviewed tools is very high, the discussion is not in deep. However, it provides an overview of the existing meta-environments and can be used as a great preliminary framework.

The five classes distinguished in this paper are environment frameworks, customizable environments, process modeling, process programming, and tool integration.

Environment frameworks are those that support a set of low level services. Object Management Service (OMS) is an example of a low level service that provides for persistent objects and relationships as oppose to the traditional filesystems. User Interface Service (UIS) is another example that provides mechanisms for defining user interfaces and associating environment objects. The Portable Common Tool Environment (PCTE) is a typical environment framework with an Entity-Relationship-Aggregate (ERA) model-based OMS and a set of user interface primitives that can be used as a basis for integrating tools as part of the development of an environment [83].

Customizable environments provide a high level and fixed core of services and sometimes allow the users to extend the core capabilities. The main difference between this class and the frameworks is that they model a small portion of the environment and allow for customization of that portion while fixing the rest. For example meta-programming environments assist in creation of parsers and related tools which manipulate a particular language. Most MetaCASE tools are hybrids of the frameworks and customizable environments. They offer low level services, such as the ER-based OMS of Metaview [78] or the graphical UIS of Toolbuilder [2], as well as customizability of a large portion of the environment.

Karrer and Scacchi believe that the above two classes adopt particular software processes and consequently cannot be useful for other processes. The process modeling class of meta-environments attempts to overcome this problem. Here the provided software process model can be instantiated to specify

[5] Also Known as meta-environments [25, 40] or environment generators[89].

the activities, developers, resources, artifacts, and their relationships which together forms the environment. We believe the generality of such a model reduces the possibility of providing focused and useful tools.

Process programming uses a programming language (like Ada) to describe the processes that form the capabilities of the environment. It seems the expressiveness of a programming language in modeling of the software process maybe of the concern. Perhaps it is most useful in providing programming support environments.

The final class, tool integration, deals with integrating various environment tools. This is done either by adopting a set of standards during development of new tools or by integrating existing non-standard tools.

1.4.2 Martiin et al's Review

A more focused and detailed comparative review of meta-technology is the research result of Martiin et al from the MetaPHOR project [46, 49]. The acronym MetaPHOR stands for Meta-modeling, Principles, Hypertext, Objects, and Repositories. These words are used to show objectives of this project which is to build a configurable CASE tool by applying object-oriented modeling philosophies in a distributed computing environment and using modern hypertext-based user interfaces. The result of this project is the MetaEdit MetaCASE tool [53].

Martiin et al's comparative review of MetaCASE tools is based on a framework addressing two issues. The first one is the properties of MetaCASE tools and the second one is their effectiveness and usability. The properties of MetaCASE tools are divided into linguistic, functional, and mechanisms. Linguistics refer to the description languages used in defining conceptual structure of models and their textual or graphical representations. Functional relates to the essential data management facilities like query and report definitions. Mechanisms cover user interface, data communications, and operating system issues. Effectiveness and usability of MetaCASE tools are studied in the meta-model, user interface, and design tasks and functions areas.

The selection of the MetaCASE tools for the study is based on classifying the tools according to the style of customization. Four styles are identified. Database oriented tools use a meta-language to define the methodologies; like MetaPlex [12], Metaview [78], and QuickSpec [62]. Interface oriented tools have generic graphical notations for building the environment; like RAMATIC [5]. Extension kits involve extending an existing tool to include new meta-languages; like Excelerator and its Customizer [17, 26]. Finally, knowledge oriented tools like ConceptBase have a meta-model based on logical rules [66].

Martiin et al discuss the first three of these categories and review a representative tool for each of those three. They compare them based on properties and effectiveness as outlined before. Although a good deal of detail is put

into each tool and an example environment has been developed by all three selected tools, only three tools are covered. In our opinion a detailed study some of the tools not covered in this review is necessary and timely.

2 Framework of Our Review

As discussed in the previous section, existing reviews provide some very helpful preliminary information about MetaCASE tools. However there are still many research issues that need to be addressed. There has been very little detailed study of the existing tools. The one detailed study that we know considers the tools from the user tasks and accomplishments point of view (Martiin et al's review).

Our approach in examining MetaCASE tools is based on studying their system architectures. We believe this study would be beneficial for researchers of this field as well as the stake-holders from industry. Analysis of the components of MetaCASE tools and their interconnections can help us identify the common parts and recognize the weak spots. It can be useful in evaluating the performance of the tool and isolating the bottlenecks. In order to do this we will introduce a typical and general architecture for MetaCASE tools which is our preliminary modeling attempt. It is provided to be used as a common architectural framework in our study of different tools.

2.1 Typical Architecture of MetaCASE tools

Figure 7 shows our preliminary typical model of the architecture of MetaCASE tools. In this figure the core components are represented by boxes with solid lines. These components are parts of almost all MetaCASE tools. The other components are the possible additions and variations. As an example a tool maybe distributed with more than one data store or it may support group-work with more than one user. The interface between these components is fuzzy and depends on the individual MetaCASE tool implementation. In an ideal open architecture, each component should be replaceable by other ones. Hence there maybe different components that are interchangeable. An examples is the "Other User Interface" which may substitute the original one based on the application type or the user preference. In the following sections we will describe the typical components of MetaCASE tools.

2.1.1 Data Storage, Access and Descriptor Facilities

The storage of data involves the low level data management issues such as file locking, concurrency control, data sharing, and mapping of data from physical to conceptual level. A data management system is often used to control the

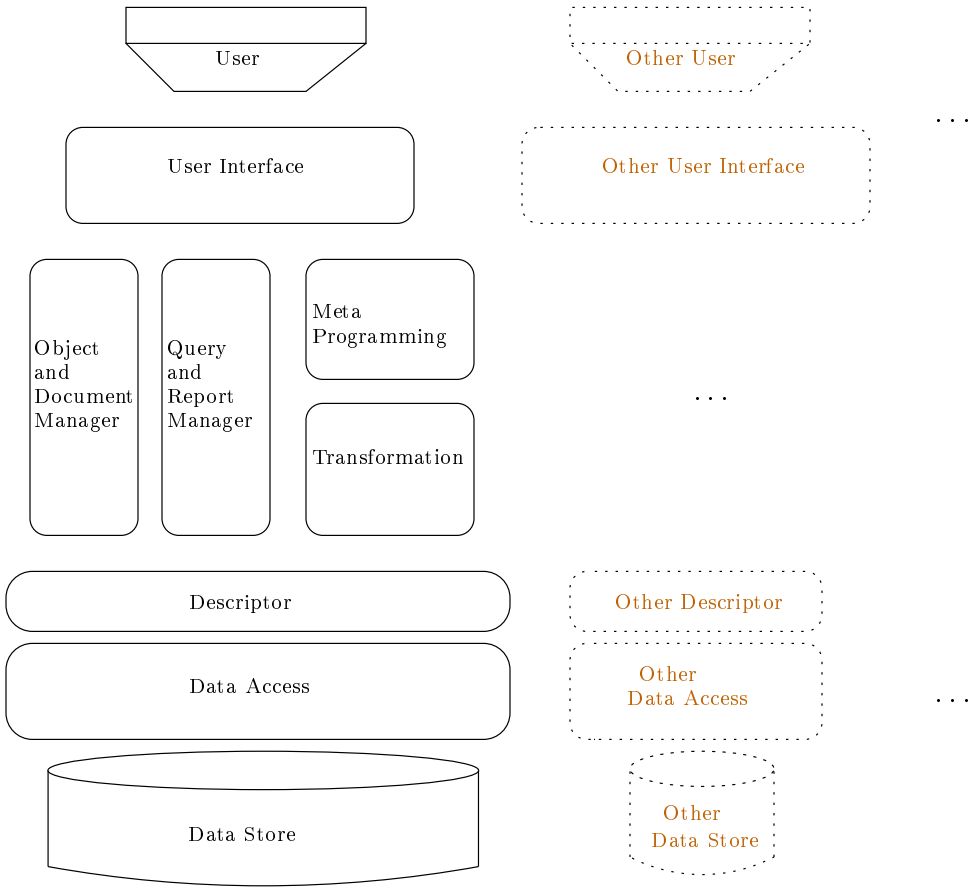

Figure 7: Components of a Typical MetaCASE Tool

access of data in the storage and provides functionality such as locking that would make the tool a multi-user one. Another issue is the possibility of network computing and distributed databases.

The data access allows the modeling and manipulation of the data at the conceptual level. By modeling of data, we mean using data structures to specify the methodology and the actual software systems. Descriptors are the models of the methodologies which are also called meta-models. Often one or more meta-modeling technique(s) are used in a MetaCASE which we can categorize into the following three types:

1. **ER-based:** Some MetaCASE tools use traditional entity-relationship modeling techniques, where concepts are represented as entities with relationships among them. This allows easy modeling of most of the concepts in structured analysis and design methodologies, but requires extensions for hierarchy of entities and their graphical representations. However, even with the recent extensions this modeling technique fails to provide the semantic power needed for modeling more complex concepts

(as discussed in section 3).

2. **OO-based:** Other tools use more recent object-oriented modeling techniques. This differs from ER-based techniques in allowing object instantiation as well as the use of object modeling notation. This permits easy modeling of most of the concepts in OO-based methodologies but requires extensions to include concepts such as aggregation, generalization, and graphical representations. The existing OO-based meta-models often contain too many details and lack a clear and predefined structure and functionality (details in section 4).

3. **Graph-based:** The third meta-modeling technique refers to graphs that are hierarchical and formal (set-theoretic or logic-based). They often have visualization and querying aspects with associated mechanisms and languages. Since the concepts of methodologies are often graph-like, graph-based meta-modeling techniques match the natural form of these concepts and, therefore, are more expressive. In addition, they are formal and include visualization and querying languages providing predefined structures and functionality with more semantic power (described in section 5).

There are further issues, like the evolution of the software system and the reuse of the components, that need to be addressed by any MetaCASE tool. These issues arise during the customization of the CASE tool and more importantly while using the CASE tool. Often they are methodology dependent. Therefore, the meta-modeling technique must provide the appropriate capabilities. Evolution refers to concepts such as re-engineering, reverse engineering, and design recovery of the existing software systems. Reuse is concerned with providing knowledge storage and analysis facilities including libraries of GUI or program-code components.

2.1.2 User Interfaces

User interfaces make up the front end of the MetaCASE tool. Often a combination of graphical and textual interfaces are used as a bridge between the system and the user. Since many of the concepts in software engineering involve graphical elements and since use of these elements helps understandability, by reducing the complexity, there is a strong need for a usable graphical interface. However, some of the concepts maybe better and faster understood and manipulated using a textual interface. Hence a combination is often desired.

Another factor to consider is the type of user. MetaCASE is used in two different levels as tools for building information systems as well as software systems. Information engineers in one level build meta-models of methodologies and produce a customized CASE tool. The customized CASE tool is then

used by developers in building software systems. Users in these two levels may have different computer knowledge and expertise. A good user interface should provide tools according to the knowledge and expertise of its users.

A current popular tendency is towards a common user interface. This is based on the assumption that information engineers or CASE customizers and the software engineers or CASE users are the same people. It is believed that a good approach is to let the developers customize their own CASE tools and use the customized tools to build the software systems.

In any case, the user interface needs to provide consistent facilities to reduce the learning curve. A user should be able to find and perform the common commands in a common and consistent way throughout the MetaCASE tool. Differences may arise based on the requirements of different methodologies but these should be understood by the user. We believe this aspect of MetaCASE is very important. A consistent and easy to learn user interface can mean customizing and quickly using customized tools in shorter times which means increased productivity.

2.1.3 An Object and Document Manager

The object and document manager is the direct back end of the user interface. With this facility, graphical or textual objects are assembled for viewing or manipulation by the user through the user interface. Interaction with the user is the responsibility of the user interface but the management and organization of the artifacts is the responsibility of the object and document manager.

Documents are the artifacts created during software engineering process. These range from the data flow or the module specification diagrams to the test plans, user manuals, and the source code. Organization and management of such documents involves gathering all the related information from the data store, interpreting and applying the enforced policies of the methodologies from the descriptor, and allowing user customizations. Users often outline a document and the tool would search for the related artifacts, collect them while enforcing the descriptor's rules, create the desired format by applying possible transformation techniques, and display or output the result in the appropriate format.

Perhaps a form of process support falls within the responsibilities of this component. This requires a way of defining the process involved in building software systems as prescribed by each methodology. In a sense the process maybe embedded in the descriptions of methodologies which would make it a part of the descriptor component. A simple example of this type of functionality is the order in producing documents that maybe enforced during the engineering of the software. Supporting such a functionality requires an active and triggering type of mechanism that interacts with the user and manages the process of the software production. In a more general sense software process

support allows guidance and coordination of various activities and management of the produced documents.

2.1.4 A Query and Report Manager

This component deals with user queries and the retrieval of the respective information. Queries vary from simple requests of information about states of certain documents to more complicated inquiries about the methodology. Reporting the results of queries may involve searching and navigating around the documents, performing logical and arithmetic calculations, and displaying the results.

Queries may involve requesting information about the methodology during the customization of the CASE tool. Often the CASE customizer would inquire about the consistency and correctness of the meta-model with respect to certain logical rules and conditions. This may involve having a meta-model which is mathematically manipulable.

Queries may also be about the software systems under design. The simplest query would be to ask about "a certain input item" or "object" of the system. This involves searching and navigation through the documents. A more complicated query may deal with checking "if the system can be in such and such state at the same time". This involves a simulation of the dynamic behavior of the system and a search through the possible states. Perhaps some of these queries are quality assurance issues such as consistency, completeness, and conformance to the rules of the adopted methodology. These facilities identify errors and problems in the form of reports.

2.1.5 Transformation and Meta-programming Tools

These tools refer to the activities involved in "lower CASE". The ideal is to have the MetaCASE automatically transform the requirement specification onto the source code. Transformations should be mathematically provable, in which case, the source code would provably satisfy the requirements. However, in reality this is not always the case. Very few MetaCASE tools offer automatic code generators and those who offer require extra information from the developers to be inputed and checked manually at earlier stages.

According to the used methodology, transformation may involve the automatic generation of one or more of the documents (like the module specification document) from the previous ones (like the data flow and state transition documents). Often the level of the automatic generation is dependent on the level of the formality of the originating documents. Methodologies prescribing a formal specification of the design would often provide transformation mechanisms that, if employed, can lead to the automatic generation of the code.

Meta-programming often involves providing the methodology prescribed programming support environment from the choice of the language to the parsers, compilers, and debuggers specific to that language. In most cases automatic transformation mechanisms provide the programmer with the module interfaces and program headers. Then according to the specified behavior the final coding is done and tested against the requirements,, all within the guidelines of the specific customized methodology.

2.2 Categorization and Selection of Tools

Based on the architecture discussed in the previous section we can now classify the MetaCASE tools and select representative ones for the study. To classify we will use the underlying data structure representation and meta-modeling techniques as our basic framework of study. We believe this approach is sound and reasonable because most of the functionality provided by a MetaCASE is dependent on its meta-modeling technique(s); fundamental limitations and capabilities of a tool can be traced back to the meta-modeling characteristics. As an example, if a MetaCASE customized CASE tool does not allow modeling of the nested objects in a system then it is likely that the underlying meta-modeling facility fails to model the hierarchy of the objects. Perhaps a simple ER-based meta-modeling technique has been used.

As discussed in the previous section there are three major types of meta-modeling techniques: ER-based, OO-based, and graph-based. Almost all the tools often extend and modify these basic modeling techniques to facilitate modeling of more complicated software engineering artifacts. Examples include set theoretical concepts such as hierarchy, diagrammatic objects, and graphical management capabilities.

Our division of CASE tools, which is based on their meta-modeling techniques, differs from the existing classifications in more than one way. One classification is based on the scale of the system that the CASE tool is capable of supporting [60]. Karrer and Scacchi's approach is based on the provided services or the type of the adopted software process. That is a broad classification of more than just MetaCASE tools. In fact, in their approach MetaCASE tools are hybrids of two of the classified divisions. Martiin et al classify only the MetaCASE tools, however, their framework is based on the properties and effectiveness of the tools.

In the following three sections we will provide examples for each of the categories mentioned and select and review some representative tools:

3 ER-based Tools

Some of the MetaCASE tools that use an ER-based meta-modeling technique or an extension or variation of it are Metaview, MetaPlex, Socrates [84], Totem [79], Toolbuilder, RAMATIC [49], and Customizer. As a representative of research tools we review the Metaview system which we chose because it is a project with a history of research and a wide variety of publications. We examine the Toolbuilder system which is one of the most popular commercial MetaCASE tools in the market and has not been reviewed before. A review of the last two tools can be found in [49].

3.1 Metaview System

Metaview project is a joint effort between the universities of Alberta and Saskatchewan in Canada and dates back to McAllister's 1988 Ph.D. Thesis 1988 [51]. The result of this project is the Metaview MetaCASE tool which is primarily a research tool. Currently, this research continues in areas such as incorporation of the "methodology knowledge" into MetaCASE tools and better representation of "aggregation" in the meta-modeling techniques employed by Metaview [77].

Figure 8 outlines the architecture of the Metaview system using a data flow diagram. Based on data flow diagram conventions, boxes are labeled "P" for processes, "S" for data stores, and no labels for the external users and developers interacting with the system. The arrows show the flow of information between the boxes. This diagram divides the system into three levels of description (meta, environment, and user) which we have described in section 1.3.

In the meta level there are two types of softwares developed by two types of developers. Metaview definer describes one type of software which forms the Metaview software library (S1) and tool definer specifies the required tools to form the Metaview tools library (S3). The main difference between these two types of software is that the programs of Metaview software library (S1) are not directly visible to the end user. In fact, in the environment level they are used by the method definer to form the methods library (S4). Furthermore the software process engineer makes use of all three libraries (S1, S3, and S4) to configure the system and produce the customized CASE tool. The system developer in the user level uses this tool to produce software systems (S5). In the following subsections we will describe the components of Metaview system based on our architectural framework.

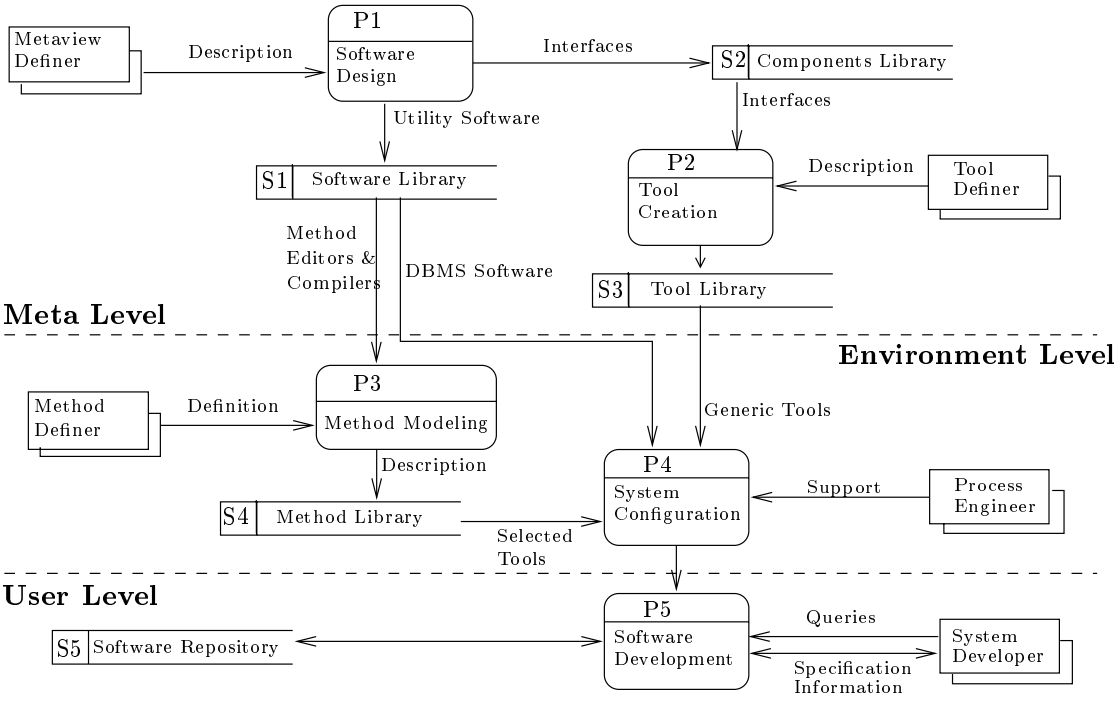

Figure 8: Metaview Architecture

3.1.1 Data Storage, Access and Descriptor Facilities

The meta-model defined for Metaview is EARA/GE (Entity - Aggregate - Relationship - Attribute with Graphical Extensions). It is designed to describe various software development methodologies. The entities, relationships, and attributes are the familiar concepts from the ER modeling technique. However, Metaview extends these concepts to include specialization, aggregation, and some generalization elements. These extensions allow data abstraction and modeling of the hierarchical nature of the artifacts and components of software systems during the engineering process. This permits representation of heterogeneous collection of entities and relationships as a single entity. Furthermore a graphical extension to EARA model supports definition of graphical representation of software objects.

Among the methodologies modeled in EARA/GE, we can name Structured Design [91], Higher-Order software [50], and recently the OMT methodology [68, 93]. The concepts of these methodologies are defined using the *Environment Definition Language* (EDL). This language allows modeling of many of the concepts present in the methodologies in a textual form but lacks any graphical formalism. In defining an environment using EDL there is a need to define constraints for the objects of the EARA. The constraints enforce a completeness and consistency checking on the objects. They are defined using

the *Environment Constraint Language* (ECL).

Metaview is able to model a complex OO methodology (OMT) however it has limitations. These limitations are most visible when fully modeling a complex environment. Among these limitations is the lack of possibility of creating two relationships with the same participants. Furthermore the aggregation mechanism which is the main feature of the EARA fails to work well with the graphical extension.

Data storage facilities of Metaview are various libraries of components, tools, softwares, and methods. The components and tools libraries (S2 and S3) support the user interface facilities. The software library (S1) provides the database management and utility facilities to be used by the software process engineer. The methods library (S4) contains the descriptions of the software development methodologies. The software repository (S5), also referred to as specification database, contains the developed softwares. At the user level the *database engine* is a prolog-based database management system, which manages the specification database.

3.1.2 User Interfaces

Metaview system is an X-windows based multi-user system. Besides the meta and tool definers, there are three types of users, not necessarily distinct, with three different interfaces of Metaview.

At the environment level the method definer interfaces with the system by providing the EDL and ECL source codes and creating the graphical objects using a primitive graphical editor. These codes are compiled using the EDL and ECL compilers. Then tables describing definitions of the objects and a set of predicates enforcing constraints are created. Currently the compilers can not handle the graphical aspects of the objects including the graphical constraints. Therefore creating graphical tables and constraints is done manually. Further research in this area is under way.

Configuration of the CASE tool, to be used by software developers, is done by the software process engineer who selects one or more methods and configures the tools. Currently specialized tools are configured but the only universal tool, "*Metaview Graphical Editor*" (MGED), stays the same. It is anticipated that other tools maybe added later. The database engine is also customized and a project daemon is created to handle consistency and completeness checking. This configuration requires a large amount of knowledge about the methodologies, tools, and the system under development. All the work is done manually and there are no help systems.

At the User level the project daemon initializes and controls the server. It allows multi-user access and arranges the appropriate locking mechanism. The developer accesses and modifies the specification database via an interface based on the MGED tool. This tool allows graphical editing of the objects of

the specification database. Operations such as creation, deletion, and modification of objects, edges, and icons are supported using a variety of graphical editing facilities.

3.1.3 An Object and Document Manager

Tool components and Metaview tools libraries (S2 and S3) are the back-end libraries to the user interface facilities of Metaview system. These libraries are collections of C++ routines and interfaces used in building the MGED graphical editor.

One of the capabilities of Metaview at the user level is to allow automatic transformation of objects between two supported representation formats: textual and graphical (*local transformation*). The preliminary efforts in this area are towards automatic transformation of the diagrams created using MGED to Prolog-form compatible for the Prolog-based specification database. Further research in this area is required.

In another sense, *global transformations* of documents between different formats, at different stages of development, or even between different methods are supported. This is done using a language called the *Environment Transformation Language* (ETL) which we will discuss in section 3.1.5.

Among the software in the software library is a *Project Daemon*, which is a server program that provides a uniform interface between the tools and the database engine. This program is responsible for handling the concurrent accessing of the database from the tools. It also manages the specification documents and enforces consistency.

Metaview has an executable process model specification mechanism. This model is based on an active database model. A special language, which is designed and hoped to be implemented in future, allows describing the process models, after which, automatic triggering rules act on the database and provide the necessary interaction with the user [6].

3.1.4 A Query and Report Manager

The methodologies in Metaview are stored in the methods library. In defining these methodologies there are no user supporting systems and no querying is possible about the correctness and consistency of the methodologies.

The specification database is managed by a Prolog-based database that allows simple queries about the states of the objects. Often these queries involves searches and tracing of the objects and requesting information using the embedded metrics. These metrics must be defined explicitly at the environment level, using the ETL together with the definition of each environment. As an example, metrics for DFDs and structure charts are embedded in the EDL and ETL of those environments [8]. They are often simple count functions that

give the number of input and output data elements, processes, terminators, etc.

The reporting is done using screens that display the metrics for each object or group of objects. Furthermore rules specified in ETL, allow the system to calculate certain metrics and, based on the result, give some assistance to the user. As an example the depth of the decomposition in the DFDs can be analyzed based on the number of the data elements at each level. There seems to be no other reporting facilities available at this point.

3.1.5 Transformation and Meta-programming Tools

Metaview supports semi-automatic transformation of formal documents from one format or environment (like DFD of structured analysis) to another format or environment (like structure chart of structured design). This form of global transformation is possible since both environments are modeled with the same formal language (EDL) and the rules of transformation are also expressed formally using a set based language called ETL [7].

Explicit definitions of the specific ETL for a particular transformation take place at the environment level. The developers later specify a software system in one environment and use the ETL definitions to automatically transform them into the first-cut of that system in the other environment. Finally they must manually improve, optimize, and complete the first-cut to a full and valid system in the other environment.

Metaview focuses on the "upper CASE" and provides tools for the analysis and design stages of the software engineering process. It does not provide any means of transforming the design into source code, meta-programming, or any other "lower CASE" activity.

3.2 Toolbuilder System

Toolbuilder is a commercial MetaCASE tool built by IPSYS Software in UK. Currently this tool is used by many researchers around the world in building experimental CASE tools. Research is on the way to extend the tool in automating more of diagram editing and correctness checks [28].

Toolbuilder has a method specification capture component (METHS) that captures the underlying meta-model of the methodologies including the language of the diagrams and the input or output structures. These specifications are then transformed into parameters for generic tools and mechanisms which form the run-time component. This component interacts with the user, capturing the specification of the software, according to the rules of the parameterized methodology [2].

It has been difficult to obtain technical detail about Toolbuilder since it is a commercial tool. However, there are documentation prepared by academics

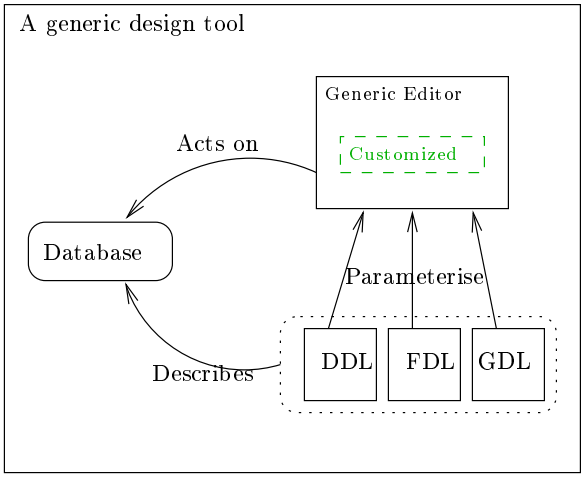

Figure 9: Components of a Generic Tool

using Toolbuilder that provide some technical detail [28]. Based on these documentation, Toolbuilder seems to be a collection of integrated and generic tools and function libraries. Toolbuilder has two editors (diagram and text editors) and allows building of the corresponding tools. Figure 9 shows the constituent parts of a diagram editor (or design editor). Here DDL describes the underlying data structure, FDL provides an interface, and GDL describes the symbols and graphics. These descriptors parameterize the design editor and define its allowed behavior. With this architecture the design editor stays as a generic tool and is capable of providing some functionality. However often there is a need for more specific capabilities. DDL, FDL, and GDL describe what is allowed and not how a desired action is to be performed. To make the tool method-specific (capable of providing specific functions), Toolbuilder allows the user to write customizer functions and link them to the design editor. In this case the design editor is parameterized by the descriptors and has an specific behavior defined by the linked functions. At a lower level these functions can be C functions.

3.2.1 Data Storage, Access and Descriptor Facilities

The meta-modeling technique defined for Toolbuilder is based on the realization that diagrams used in methodologies generally take the form of a directed network of nodes and links. The nodes are modeled by entities and links by relationships. Diagrams also contain labels with values that are modeled by attributes and their values. The symbols and graphical styles are recognized as lexis of a graphical based language. The legal uses of symbols are the syntax and the legal relationships among nodes and links are the semantics. Similar principles apply to forms, structured text, and matrices. Hence a language is

defined to capture and describe the methodologies.

Data description language (DDL) is the language used to describe the schema of the data. Schema is meant to be the definition of entity types, their attributes, and relationships among them. The database or the data dictionary of Toolbuilder is produced by a compiler that takes the schema created by DDL definitions. This database has two tiers: the directories and file level, and the file content level. There is also a distributed OO database management system which is proprietary to the IPSYS company and is used to manage the database. This system provides the necessary locking and concurrency control.

The meta-model of Toolbuilder covers more than just the simple entities and relationships. In this model entities have types that determine the allowed attributes. Attributes also have types that maybe printable, like strings and integers, or an entity by themselves. These latter types are known as the links or the relationships between entities. There are constraints on the types and cardinality of the entities and their attributes. As an example an attribute of an entity maybe of the same entity type and it maybe single or multiple valued.

This meta-model supports hierarchy of the entities where sub-types inherit the attributes of their parent type. It is possible to define derived multiple valued relationships as the collection of the values of a group of other relationship(s), which is the familiar notion of the aggregation. In addition recursion is also supported which defines a derived relationship by a recursive function over another relationship. This meta-model is active in that triggers can be associated with events applying to attributes and relationships.

In addition, the graphical information about a diagram, like the various types of shapes, lines, texts, matrices, maps, and the relationship between them is described using the graphical description language (GDL). On the other hand, textual editing tools are provided with the description of how they are to be edited using a layout language (LL). Yet at run-time most of the functionality of tools are provided by the customizer functions which we discussed in previous section.

Toolbuilder has been used to encode SSADM which is considered to be a complex methodology. In addition it has been used in many MetaCASE related research projects including teaching, reverse engineering and re-engineering [3]. Our general impression is that Toolbuilder provides a very effective mean for capturing data model of methodologies and creating the appropriate diagrams and text editors. Often this is most effective when the engineer is very familiar with the domain and yet very ineffective when the users are not experts.

3.2.2 User Interfaces

The user interface of Toolbuilder is OSF/Motif based which is usually used in the Unix environment. However Toolbuilder has addressed portability across operating systems and hardware platforms. As an example, efforts are on the way to port Toolbuilder into Linux environment. The customized tools are often single-user tools, however, Toolbuilder provides facilities for teamwork.

The interaction styles of the user interface (windows, menus, buttons, etc.) and the layouts are described by a format description language (FDL). FDL parameterizes the tools by associating specific actions with the creation and selection of an interaction object like a menu or an icon. The interaction is through the design diagram and structure text editors which capture both the diagrammatic and textual aspects of the methodologies or the designed software systems.

During the specification of the methodologies there seems to be no help available for the CASE tool customizer. However with the specification, there are (not very explicit) facilities that would allow building of help systems, metrics, and guidelines to be used by CASE users. In general, a conclusion in using Toolbuilder by Ferguson is the extreme complexity when extending the functionality of the tools beyond the default operations which, together with poor documentation, requires a lengthy learning curve [28].

3.2.3 An Object and Document Manager

Toolbuilder manages user interaction with a user interface management system. Initially a log-on facility is responsible for setting-up the environmental variables used by the tools, compilers, and the database. Once logged-on, the user interface manager controls the interaction between the tools, compilers, and the database. Compilers are the back-end to the languages of the Toolbuilder (DDL, FDL, LL, and GDL). These languages parameterize the run-time component and influence the METHS component that captures different aspects of the methodologies.

Toolbuilder addresses the user's view of the underlying data model in its frame model. Frames are collections of graphical or textual objects with images on the screen and associated actions. The actions maybe default functions, like cut-and-paste, or more specific functions like navigation. There is a root frame which is invoked by the run-time component and determines the presentation of the information to the user. FDL, LL, and GDL scripts are all generated within the frames for use by the run-time component.

Data consistency among different representations is said to be the result of using a uniform meta-modeling technique in the database. However, as a result of current research, high level manipulation functions have been developed which maybe used recklessly and introduce inconsistencies. Hence there is a

need for a mechanism that would check the correctness of diagrams.

It is claimed that Toolbuilder has been used in building a management process support system. We do not have documentation of this, however, the database of Toolbuilder seems to have the capability of supporting automatic triggering rules. These rules can be used to define events and actions required in supporting a process that would be executable at run-time. This capability is implicit in the meta-model.

3.2.4 A Query and Report Manager

It appears that Toolbuilder does not provide a complete, clear, and high level querying facility. However the current stage of Toolbuilder's functionality is not clear and therefore passing a judgment seems unfair. Based on the documentation available to us, there are functions developed for accessing the database (based on the attribute values) and visualizing the data (using links and sequences). This is done by queries defined by the CASE tool customizer to be used by CASE tool user.

Toolbuilder allows generation of reports with structures defined by an specific sub-language. Once the report structure has been customized, the output generation aspect of Toolbuilder uses the design and structure text editors' off-line facilities to produce the appropriate output. These outputs maybe for the users or for the further stages of the process. The customization of outputs takes place in the following four stages: first by controlling the source of the information which is a crude form of integration, second by controlling the structure which includes scripts for generating particular representations of data, third by controlling the logical appearance at the run-time, and fourth by controlling the physical appearance directly or using a publishing system.

3.2.5 Transformation and Meta-programming Tools

Toolbuilder uses an integrated and shared data system between its tools, and allows transformations between textual and diagrammatic formats of data. This is like viewing the same data using different tools or in different environments. Furthermore, the report generating sub-language of Toolbuilder and the corresponding run-time components are used to define specific formats for the outputs. This can be done between different stages of the process.

Toolbuilder provides facilities for "lower CASE", including code generation, configuration management, and version control. This is based on its open architecture which allows third party component integration, specifically document processors and code generators. However we have not been able to obtain any documentation describing experiences in these areas.

4 OO-based Tools

Examples of tools that use an OO-based or an extension of this meta-modeling technique are MetaEdit, QuickSpec [49], Phedias [86], and ConceptBase [37]. QucikSpec has been reviewed by Martiin et al. Phedias is a fairly recent attempt in MetaCASE technology but in many ways very similar to MetaEdit and RAMATIC. ConceptBase is basically a deductive object manager for MetaCASE tools. It amalgamates an OO-based meta-modeling technique with knowledge-based features to facilitate deductive reasoning on the data storage of MetaCASE tools. Among the tools of this type we will, therefore, examine MetaEdit which happens to be a popular commercial tool with distinct methods engineering capabilities.

4.1 MetaEdit System

The MetaEdit family of tools and in particular MetaEdit+ is one of the most popular MetaCASE tools both from the research and from the industrial usage point of view. In addition, MetaEdit+ is also a *Computer Aided Method Engineering* (CAME) tool. Although all MetaCASE tools provide means of methodology specification, MetaEdit+ claims to be a CAME tool since it also provides flexible means of method management, integration and reuse. This tool is the result of a joint research project between Jyväskylä and Oulu universities in Finland. This project is named MetaPHOR[6] with an estimated cost of about 600 000 US-D within 20 man years. This research is much focused on meta-modeling theory and its applications to method engineering.

Considering our discussion about the three levels of MetaCASE modeling in section 1.3 we can discuss the corresponding three levels in MetaEdit+ [76]. As a MetaCASE tool which is used to build CASE tools, the three levels are similar to our model in figure 6. However as a CAME tool the three levels operate one level "higher" than in the MetaCASE tools. Therefore the highest level is called the meta-meta-modeling level where the syntax and semantics of various meta-modeling techniques are defined. The subsequent levels are the two highest levels in the MetaCASE tools. This allows definition of meta-modeling techniques and engineering of methodologies. We will refer to meta-meta-modeling as meta-modeling since the concept of modeling is the same at any meta level.

The general architecture of MetaEdit+ is shown in figure 10. This architecture is based on conceptual modeling principles. Hence, the repository has an associated conceptual schema and the tools resemble external views of that schema. In addition object-oriented design is used in development of MetaEdit+ which allows reuse and interoperability between tools as claimed

[6] Introduced in section 1.4.2

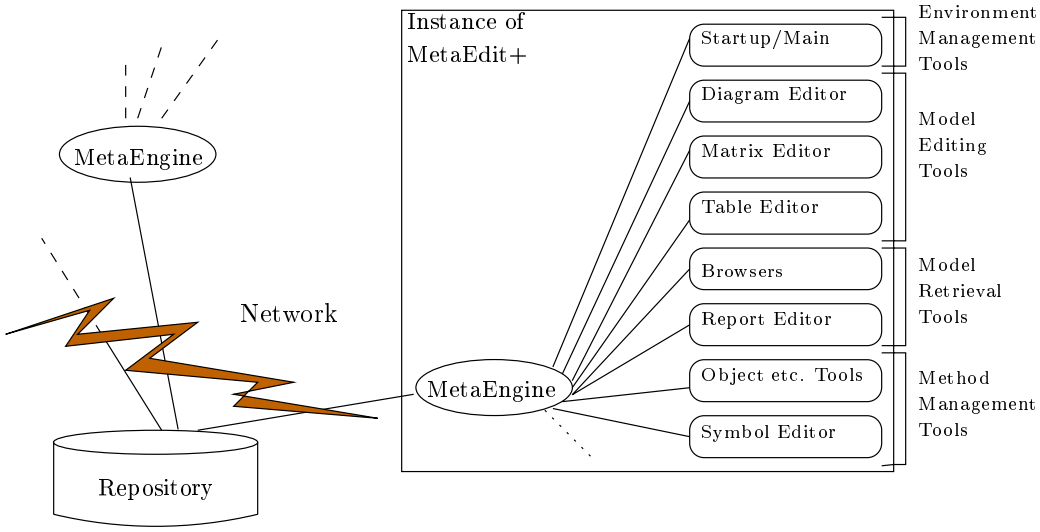

Figure 10: MetaEdit+ Architecture

by developers. The heart of MetaEdit+ is the *MetaEngine* which handles all operations on the underlying conceptual data. The tools of MetaEngine are divided into four groups: environment management tools (to manage features of the environment); model editing tools (to create, modify, and delete model instances or their parts); model retrieval tools (to retrieve design objects and their instances); method management tools (for method specification, management, and retrieval).

4.1.1 Data Storage, Access and Descriptor Facilities

The meta-model used in MetaEdit+ is called GOPRR: Graph, Objects, Properties, Relationships, and Roles. This data modeling technique is used at the conceptual level for specification of Methodologies as they are observed, interpreted, and recorded. The objects of GOPRR refer to the independent and identifiable design objects like the entities of an ER diagram. These objects often have properties that are their attributes like the name of a process in a DFD. Relationships are associations between objects that appear as lines between shapes in diagrams. Roles define specific ways that objects participate in relationships.

The concept of graph was added to the meta-model for several reasons. It denotes an aggregate concept which contains a set of objects and their relationships. This allows representation of complex objects using decomposition into contained objects. In addition this design allows different "representational" graphs (like matrix or diagram) of the same "conceptual" graph (like a DFD). Furthermore it makes possible the use of multiple methods at various levels of the project while maintaining links among them.

Other capabilities provided by the graph concept are the inclusion of generalization and specialization constructs as well as polymorphism. This last extension allows viewing of an object in one method as an object and in another method as a relationship or property. GOPRR allows method integration in such a way that objects, relationships, and roles can be reused and properties can be shared. It provides rules for checking model integrity, consistency, and completeness which is done by defining constraint rules that are attached to the properties of a given object, role, relationship, or a graph.

MetaEdit+ provides support for OO analysis and design methodologies such as OMT and Booch as well as structured analysis and design methodologies such as Yourdon SA/SD. Developers claim that their system provides one of the largest selection of method supports. An advantage of this system is the capability to change the description of the methodology even while it is in use. These changes are said to be reflected on the models immediately. The major focus of this tool is on capturing the data about the methodology. Little work is done on ensuring correctness or providing detailed analysis and simulation tools.

The data is stored in the repository of the MetaEdit+. This data includes the method specifications represented as GOPRR concepts and other information bases needed to operate the tools. Among them are the information about the symbols representing objects, information necessary for the operation of the tools such as the spatial coordinates of objects, information describing various users with their status, and finally reports and other output specifications. There is a data management system that allows data sharing and concurrency. This is achieved using a locking mechanism which admittedly reduces the performance.

4.1.2 User Interfaces

MetaEdit+ is a multiuser system with client-server architecture that is also multi-platform. Each client has an instance of MetaEdit+ with copies of all the available tools and the controlling MetaEngine. The main launcher allows only the permitted capabilities based on the login information and the user status located in the repository. As an example, a browser allows hierarchical access to the models and the meta-models stored in the repository based on the user permission. Further interfaces with the system is through the graphical or textual tools.

Specification of the methodologies is through the conceptual object type tools, such as the relationship tool, and their textual representations. This is often done by filling in a form that describes the conceptual object. A symbol editor is used to specify and design the graphical objects and their behavior while linking them to the conceptual object type. There seems to be no help available during the specification of the methodology. However facilities are

provided for defining helps, metrics, and reporting systems for the CASE users.

CASE tool users interface with the system using model editing tools such as the diagram, matrix, and table editors. These tools are invoked from a *WorkSpace* which holds information about the user and can be configured to the users likings. Added facilities by the CASE tool customizer can provide help and guidance systems that maybe triggered automatically and metrics that maybe used to produce reports.

4.1.3 An Object and Document Manager

The MetaEngine is the direct back-end of the tools used by the users to access the repository. The implementation of the conceptual data model and the operational signatures are embedded in the MetaEngine. Accordingly, tools request services from the MetaEngine but operate solely based on their own specific paradigm. In the client-server architecture MetaEngine deals with all the communication issues with the server and even between the tools. Hence, data sharing is made possible through data integration.

As discussed earlier this architecture allows transformation of objects from one format to another one. This is possible between the three supported formats: diagrammatic, matrix, and tabular. In addition, it is possible to view objects in one method as an object and in another method as a relationship or property. Although such a flexibility is most desired, it may cause problems. Among them is to ensure consistency and integrity within and between methods. This problem is the topic of one the future research directions of MetaEdit+.

The research is on the way for a flexible process support system which guides and coordinates various activities and manages deliverables in MetaEdit+ [48]. An activity in process modeling refers to an information system development or managerial task (or a composition of tasks) that uses, or produces, a deliverable or acts as a managerial event like significant dates of a project. Such a support is project tailored and involves all the managerial aspects of the project. Therefore, it requires a rule based triggering mechanism, a version control for all the deliverables, and navigational as well as viewing capabilities. An aspect, not available in MetaEdit+, is the selection assistant that guides the user in selecting the right Methodology for a project or a part of it. Ongoing research in this area is also acknowledged [88].

4.1.4 A Query and Report Manager

MetaEdit+ provides tools for editing specification of methodologies which gets stored in the repository. No support is available for making queries about the correctness of the methodology, however, its consistency and completeness is verified during the description process.

Querying the states of the software developer specified data is possible using the customized querying and reporting facilities. The CASE tool customizer can use the report editor and its procedural query and data manipulation language to generate the capability for producing textual descriptions of the models. This capability can be used by software developers in querying the models of the repository. These queries maybe simple checks about the consistency and completeness, requests about the states of the data, or more complicated SQL type queries. MetaEdit allow creation of various reports that give textual descriptions about the model of the software system under construction.

4.1.5 Transformation and Meta-programming Tools

Objects of MetaEdit+ can be viewed by different tools which act as local transformation facilities. In essence, there is only one copy of the objects which maybe viewed in different environments and formats. An example is a textual-to-graphical transformation. However, there are no additional transformation languages that could be used to transform models between methodologies or tools.

Transformation between different phases of the software engineering process is possible using the combination of the specification of the methodologies and the report generating facilities. With the help of the report editing tool, functions can be defined to take data inputs from one phase (like object descriptions) and produce data outputs in the next phase (like language dependent class definitions). Although the focus of MetaEdit+ is to act as an "upper CASE", there are semi-automatic facilities that could be used as meta-programming tools. MetaEdit+ provides predefined SQL and Smalltalk generation capabilities.

5 Graph-based Tools

This final category of MetaCASE tools is a less explored category. The meta-modeling technique of the tools in this category is an attempt to overcome the difficulties with the OO-based and the ER-based techniques. A graph-based meta-modeling technique is said to be expressive enough to model all the aspects of methodologies and yet provide clear and easy to use querying and manipulating capabilities. Examples of tools in this category are a prototype tool by IBM called 4thought [71] and a proposal level tool by JRCASE called CASEMaker [73]. We will examine both of these tools.

5.1 4thought System

The 4thought prototype is a result of the Advanced Software Design Technology (ASDT) project of the Center for Advanced Studies (CAS) in IBM Canada, which ended in 1995 [70]. The underlying theory for the development of 4thought is the Theory-Model paradigm [43, 69] which is an attempt to provide confidence about the correctness of a complex design. Of course the real solution to the problem of correctness is the "proof" paradigm which requires great effort in using formal methods and consequently is not practical.

In the Theory-Model paradigm there are theories (methodologies) that prescribe categories of design concepts and rules. A design is said to be a collection of facts that form a model for the theory. Once the model falls within a category and satisfies the rules, it is assumed valid. This approach is more practical than the formal proof approach. Yet it is still partially formal since methodologies are described using a formal specification language such as Z [21]. This paradigm provides the theoretical basis for the development of 4thought system.

It is important to notice that 4thought approaches MetaCASE from a different viewpoint. Most MetaCASE tools focus on providing means of capturing the data model of methodologies using various meta-modeling techniques. Then they focus on providing ways of customizing easy to use CASE tools to be used by the software developers. In this process, ensuring the correctness of the produced software system is secondary. With 4thought, the focus is towards verifying that a design meets the constraints of the methodology with which it was developed.

4thought is only a prototype and lacks a complete architectural design. Based on the available documentation, it is described as a collection of stand alone components. These components are integrated to provide the functionality that supports the Theory-Model paradigm. First, methodologies are formalized using various formalization techniques. Next, they are modeled, both graphically and logically, using meta-modeling techniques. Then, executables are generated using Prolog programs. Finally, software systems are specified and checked using the executable methodologies.

One of the components that is used in 4thought is a Prolog database server that contains the information captured using meta-modeling and formalizing techniques. The visualization of data in the form of graphs is through a visualization and querying system called GraphLog [15]. The textual views of the data is also available which maybe integrated with the graphical information and using the hypertext technology linked together.

5.1.1 Data Storage, Access and Descriptor Facilities

Based on the Theory-Model paradigm, methodologies are first formalized. This step is particularly important for the informal methodologies. The key goal in doing this is to ensure that there is a precise meaning for the concepts found in the methodologies. Since most of the fundamental concepts in software engineering can be described using elements of set theory and predicate logic, a formal language such as Z notation has been used for this part. Examples of modeled methodologies are JSD, OMT, and OOA [92]. It has been argued that only a subset of the predicate calculus would be sufficient to represent the elements of methodologies. Hence a visual logic programming language (GraphLog) has also been used to express typical predicate calculus concepts of methodologies graphically.

The meta-modeling technique used by 4thought is based on graphical representation of the concepts of the methodologies. As the first step, ER diagrams are used to model the primitive concepts. This meta-model provides a basis for visualizing concepts as graphs where entities are the nodes and the relationships are the arcs of the graph. This allows association of a semantic network with the meta-model. However, ER models are not sufficient to describe methodologies. Therefore, GraphLog has been used to provide more semantic power. It allows representation of hierarchy using the nesting of nodes within nodes. This is referred to as Hygraph structures that are formal, rich, and expressive semantic networks [15].

A methodology includes derived concepts that maybe expressed in terms of the primitive ones. These concepts may include logical constraints and rules of the methodology. Perhaps operations for viewing or updating the data, while preserving the integrity constraints, are required. GraphLog, as the meta-modeling technique, allows definition of concepts, constraints, queries, and update transactions. Since GraphLog is a logic programming language, it allows definition of the logical constraints and rules which maybe used in proving the correctness, consistency and completeness of the specified software systems.

The database of the facts about the methodology and the specified software systems are Prolog-based. The entities of the ER diagrams are represented as Prolog predicates. In addition, relationships, integrity constraints, queries and the update transaction operations of the GraphLog become the Prolog rules and facts. A database management system controls and manages transactions with the database but we do not expect much functionality in terms of locking and concurrency control since 4thought is only at the prototype level.

5.1.2 User Interfaces

The user interface of 4thought is mainly through GraphLog where the methodology and the software system data can be visualized, specified, and edited graphically. We are not aware of the details of this user interface but it is expected to be a standard data visualization and editing facility. Besides GraphLog the other facilities of 4thought are prototypes involving little or no user interface issues.

5.1.3 An Object and Document Manager

Initially 4thought allows the users to connect to the Prolog database and create a workspace that contains meta-model of a methodology and snapshots of any specified software system. Using the meta-model and the specific software system further Prolog programs can be invoked as helping or guiding facilities. Among these programs we can name the viewing facilities that create a snapshot of the data about a specific design. This facility is under research to provide a faster refreshing capability and more up-to-date views. There is also a Prolog program that allows verification of the correctness of a specified software system against the rules of the methodology. In addition, Prolog programs exist that help in searching the database of a particular software system. These programs are compiled with the standard Prolog compiler creating the run-time components.

5.1.4 A Query and Report Manager

GraphLog system allows inquiries about the modeled methodology, specified software system, and their consistency and correctness. These queries may involve searches through the database and computations about the states of the data. GraphLog is claimed to have a higher expressive power than SQL, in particular, when expressing graph traversal operations without using recursion. This maybe considered one of the most advantageous aspects of using a graph-based meta-modeling techniques.

There seems to be no reporting facilities available in 4thought other than what the GraphLog provides.

5.1.5 Transformation and Meta-programming Tools

There is a GraphLog-to-Prolog translator which takes the GraphLog representation of the system and produces Prolog facts and rules for the use by the Prolog-based database system. This transformation can be considered a formal transformation since the language of GraphLog is logic which can be mathematically transformed into logical statements readable by the Prolog compiler. This GraphLog-to-Prolog translator seems to have performance

problems which is most problematic for large-scale software systems. This area is considered to be an active research area by deductive database community.

4thought, at this prototype level, provides no other facilities for the "lower CASE" activities.

5.2 CASEMaker System

CASEMaker is the most recent MetaCASE tool project undertaken by the Joint Research Center for Advanced Systems Engineering (JRCASE) and Macquarie University in Sydney, Australia. This project is only at a proposal level which can hardly be reviewed in comparison with other more developed tools. However, we believe it addresses many fundamental issues about MetaCASE technology and is worth examining. In addition, this study can help us understand the current state-of-practice in building MetaCASE tools.

Motivation for the development of CASEMaker is claimed to be the demand for "better" MetaCASE tools. The "better" here refers to the capability of providing more than just the data capture which is about the only major functionality offered by tools such as Toolbuilder and MetaEdit. CASEMaker claims to be able to offer more in areas such as design simulation, metrics, transformations, and guidance services. However, this project is at a proposal level and results remain to be seen.

The proposed architecture of CASEMaker has two parts: components (customizable building blocks of the CASE tool), and assembly-customization tools (mechanisms for the assembly and customization of the components). There are also three sections recognized in CASE tools: the user interface, the design support facility, and the database. For each of these sections there are both the components library and the assembly-customization tools that are used by the CASE tool customizer to produce all three aspects of the CASE tool.

Details of the architecture is yet unknown but the existing documentations show numerous issues being addressed in the design of CASEMaker. The user interface of the CASEMaker is an aspect which has been researched, its necessary requirements have been identified, and the concept has been designed [47]. It is referred to as the *MetaDesigner* which consists of a group of generic GUI classes (engine) that maybe integrated and configured (into the builder) so that specific CASE tools can be built.

We will leave the review of specific components of CASEMaker to a later date when a prototype is available. At this point we can examine the fairly complete meta-modeling technique (*hypernode*) used in CASEMaker [72]. Similar to GraphLog structures, hypernodes are sets of nodes and edges of a graph where a node maybe a graph itself. Hence they generalize nodes as oppose to the edges (which is the case in hypergraphs). The hypernode model was originally proposed in 1990 and described as a graph-based deductive data model with a Datalog-based query language [44]. Later it was formalized using set

theory and the hypernode query language (HNQL) was described to be a high-level procedural query language [72]. Based on these descriptions, HNQL provides functions for the creation, deletion, and manipulation of hypernodes as well as declarative querying.

The hypernode model was finally extended by Louis Scott in his PHD thesis to include scripting capabilities by the addition of concepts such as the assignment statements and while loops. He argues that the hypernode model can provide support for modeling a variety of concepts in methodologies. This includes data structure representations, like the OO data, and dynamic behavior representations like the Petri-net data. To demonstrate this, a prototype hypernode-based CASE tool has been developed to simulate software behavior. We believe the efforts to show the effectiveness is a practical step towards building "better" tools.

We have provided some detail to introduce CASEMaker but any further detail without having a prototype would seem to shift us away from our purpose which is to review the existing tools. In general. this tool is a hybrid between a formal research work (like 4thought) and a commercial tool (like Toolbuilder). Its capabilities remain to be seen.

6 Other Tools and Components

There are other MetaCASE tools and Components that we have not categorized or examined in this paper. For example ObjectMaker [58], Paradigm Plus [81], and Graphical Designer [20] are commercial tools with little or no available documentation and Hotdraw, Hardy, Refine, and Goodstep [81] are merely components that can be used in construction of MetaCASE tools. Hence our review does not include them. An extension to this research is the examination of the remaining tools and the related components based on the framework built in this preliminary paper.

7 Discussion

In this section we will summarize and provide concluding discussions about the examined MetaCASE tools based on the framework of our study. We will use our typical component subdivisions and attempt to compare and analyze the advantages and disadvantages of each tool. A more detailed list of requirements based on the tabulation of the results is one of our future research goals.

7.1 Data Storage, Access and Descriptor Facilities

The Metaview system offers EDL and ECL languages for modeling data, graphical objects, and constraints. Similarly the DDL, FDL, and GDL languages of Toolbuilder are used for modeling and formatting of data. MetaEdit provides an OO modeling capability which is extended by the concept of graph (GOPRR). This graph concept differs from the graphical capabilities of Metaview or Toolbuilder. It refers to aggregation and representation of hierarchical concepts using graph structures.

With the meta-modeling techniques of Metaview or Toolbuilder some complex methodologies, like SSADM, can be modeled. In addition, with certain difficulties, OO methodologies can also be modeled. An example is the difficulty of Metaview in modeling two relationships with the same participants. The MetaEdit, on the other hand, is capable of modeling complicated OO methodologies, like the OMT, and may have difficulties with the structured analysis and design methodologies. All three of these tools have enforced extensions on top of the basic ER or OO meta-modeling techniques. These extensions provide capabilities for representing hierarchy. They have caused inconsistencies like the problems with the integration of the aggregation and the graphical extension that the Metaview system faces. In short they lack a clear and predefined structure.

The 4thought and CASEMaker systems use logic/set-based meta-modeling techniques with formal mathematical foundations and well developed modeling and querying languages. GraphLog and Hypernode have been shown to be capable of modeling "many" ("all" as claimed by developers of these meta-models) software engineering artifacts. However, much more work is needed to prove the usefulness of these tools since both of the tools in this category are in their early stages of development.

The databases of the existing and developed tools (Metaview, Toolbuilder, and MetaEdit) offer file locking mechanisms which permit concurrent access of the data. The commercial tools also provide facilities for software evolution which we will examine in future research.

7.2 User Interfaces

The customization of CASE tools requires experts when using the Metaview system, since the configuration of the tool as well as much of the work in creating graphical tables and constraints is performed manually. However, Toolbuilder and MetaEdit offer easy to fill form-based interfaces for customizing CASE tools that maybe filled by customizers or even the novice developers. 4thought and CASEMaker are only in the early stages of development.

None of the existing tools offer any kind of help to the CASE tool customizer. The primary objective of most of the commercial CASE tools, like the

Toolbuilder and MetaEdit, is to capture data and diagram formats, for which, form-based language constructs would be sufficient. Further customizations, specially in defining method specific behaviors, are possible but reduce the ease of use. For the CASE tool users commercial tools offer an easy to use interface which includes customizable help systems.

7.3 An Object and Document Manager

The process of CASE customization in Metaview is done at two stages of method definition and tool configuration. The method definition is basically writing EDL and ECL codes while tool configuration requires knowledge of the methodologies and is performed manually. CASE customization in Toolbuilder is simply data and graphical format capture using form-like specification capture facilities (METHS). The captured specification forms the parameters for the run-time component automatically. The MetaEngine of MetaEdit supports its method management tools that interact with the user in a form-like data capture fashion. The CASE tool is then configured automatically. CASEMaker approaches the process of CASE customization by providing customizable components and customization tools that interact with the user and configure the generic components. The 4thought system, in its prototype phase, does not offer any usable CASE tool customization facilities.

In terms of management process support, the Metaview, Toolbuilder, and MetaEdit tools either have an active database or facilitate definition of triggers that maybe used in defining events and actions necessary for supporting management process controls. The 4thought and CASEMaker systems are also capable of this functionality but no work has been done in this area.

7.4 A Query and Report Manager

Among the studied tools, 4thought and CASEMaker offer the most formal, clear, and functional querying facilities. The reporting facilities offered by Toolbuilder and MetaEdit seem to be most comprehensive, but the others also provide the basic capabilities. Often there are different levels of report generation controls but the format seems to be always text-based. The text provides a description of the methodologies or the specified software systems.

All the tools provide some form of consistency and completeness checking which is often at the diagram syntax level. However, none of the tools offer any formal way of proving theorems about the correctness of methodologies or the software systems specified in those methodologies. An exception is the formal approach of the 4thought system and its support of the Theory-Model paradigm, which allows some reasoning about the methodology and the specified software systems.

7.5 Transformation and Meta-programming Tools

In all of the tools a single repository is used to store the captured information about the software systems. This allows viewing of the same data in different formats. All of the tools offer capabilities, often as a side effect of their querying and reporting facilities, for defining transformations between phases of the software process. However, only Metaview provides a semi-automatic mechanism, through ETL language constructs, that maybe used to define mathematically based transformations.

The need for some form of meta-programming capability has been acknowledged by most of the tools. Some tools, like the Metaview and CASEMaker focus on the "upper CASE", at this point and provide no facilities for "lower CASE". Others, like the Toolbuilder and MetaEdit, provide at least a basic and semi-automatic code generation capabilities.

8 Conclusion

In conclusion, we would like to emphasize on the importance of MetaCASE technology. It has been observed that CASE and consequently MetaCASE tools are not widely used in the industry. Investigation of the underlying reasons reveal that the cause is the high cost of adopting CASE tools. In addition, the ease of use is a major factor in successful use of CASE tools. MetaCASE tools offer solutions to some of these problems however there are many open problems in this area that needs to be addressed. These open problems can be summarized by the following three items:

- **Functionality in specifying the methodologies:**
  - The meta-modeling techniques have unclear structures with inconsistencies caused by various extensions (except CASEMaker).
  - They are often limited to data-capture and provide little or no means of specifying the process or the guidelines prescribed by methodologies.
  - Methodologies are captured informally and their correctness is not ensured (except 4thought).
- **Functionality in using the customized CASE tools:**
  - Although most of the tools allow capture of the dynamic behavior, they do not provide facilities for simulation/animation of this behavior (except CASEMaker). This capability would prove useful in ensuring the correctness of the captured specification of the system.

  - Research is needed to provide a better support for the entire software process.

- **Usability of the tools:**
  - During methodology specification and CASE tool component configuration there seems to be no help or guidance available.
  - Most commercial tools provide an easy to use interface for the CASE tool user. However more guidance can be used in recommending and navigating between different methodologies depending on the application.

The contribution of this paper is the introduction of a framework for studying MetaCASE tools and grouping of the existing tools. Based on this framework, we have examined some representative tools. It has lead to the identification of the problem areas which we have summarized in this section. We plan to formulate a list of requirements which will lead us towards the proposal of a MetaCASE tool that satisfies them. The most essential part of a MetaCASE tool, as identified in this study, is the meta-modeling technique and the underlying database which we hope to design and prototype in the nearest future.

References

[1] Ada. Conference proceedings. In *2nd International Conference on Ada Application and Environment*. IEEE Computer Science Press, 1986. Standard Reference.

[2] A. Alderson. MetaCASE technology. In *Proceedings of European Symposium on SDE and CASE Technology*, Lecture Notes in Computer Science, pages 81–91. Springer-Verlog, June 1991.

[3] G. Allen and A.R. Jackson. MetaCASE technology in the support of information systems teaching. In *Proceedings of First International Conference on MetaCASE*. Sunderland, UK, 1995.

[4] M.J. Aslett. *A Knowledge Based Approach to Software Development: ESPRIT Project ASPIS*. North-Holland, Amsterdam, 1991.

[5] P. Bergsten, J. Bubenko, R. Dahl, M. Gustafsson, and L.A. Johansson. RAMATIC - a CASE shell for implementation of specific CASE tools. Technical Report T6.1, TEMPORA Project, SISU, Stockholm, 1989. Cited From [49].

[6] G. Boloix, P.G. Sorenson, and J.P. Tremblay. Process modeling using a Metasystem approach to software specification. Technical Report TR 92-11, Department of Computer Science, University Of Alberta, Canada, 1992. Cited From [8].

[7] G. Boloix, P.G. Sorenson, and J.P. Tremblay. Transformations using a metasystem approach to software development. *Software Engineering Journal*, 7:425–437, 1992.

[8] G. Boloix, P.G. Sorenson, and J.P. Tremblay. Software metrics using a metasystem approach to software specification. *Systems Software*, 20:273–294, 1993.

[9] G. Booch. *Object-Oriented Analysis and Design with Applications*. Benjamin Cummings, Redwood City, 1993.

[10] J.Jr. Bubenko. Selecting a strategy for computer-aided software engineering (CASE). Syslab Report 59, University of Stockholm, Sweden, June 1988. Cited From [49].

[11] R.N. Charette. *Software Engineering Environments Concepts and Technology*. McGraw-Hill, 1221 Avenue of Americas, New York, 1986.

[12] M. Chen and J.F.Jr. Nunamaker. MetaPlex: An integrated environment for organization and information systems development. In *Proceedings of 10th International Conference on Information Systems*, pages 141–151. ACM Press, New York, 1989. Cited From [49].

[13] P. Chen. The Entity-Relationship model towards a unified view of data. *ACM Transactions on Database Systems*, 1, March 1976.

[14] J. Chikofsky. Software technology people can really use. *IEEE Software*, pages 8–10, March 1988.

[15] M.P. Consens. Creating and filtering structural data visualizations using Hygraph patterns. Ph.D. Thesis, University of Toronto, 1994.

[16] M. Crozier, D. Glass, J. Hughes, W. Johnston, and I. McChesny. Critical analysis of tools for CASE. *Information and Software Technology*, 31:486–496, November 1989. Cited From [49].

[17] Customizer. Reference guide. Index Technology Co., Cambridge, USA, 1987. Cited From [49].

[18] M. DeBellis. The knowledge-based software assistant program. Notes Form Lectures in Stanford University. Anderson Consulting, Palo Alto, USA.

[19] T. DeMarco. *Structured Analysis and System Specification*. Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

[20] Graphical Designer. Reference guide. Advanced Software Technologies, Colorado, USA, 1996. `http://davinci2.csn.net/~jefscot/index.html`.

[21] A. Diller. *Z: An Introduction to Formal Methods*. John Wiley and Sons, Chichester, England, 1990.

[22] M.B. Dixon, J.F. Coxhead, and E.A. Dodman. MetaCASE and audit: Automated generic quality assessment. In *Proceedings of First International Conference on MetaCASE*. Sunderland, UK, 1995.

[23] A.L du Plessis. A method for CASE tool evaluation. *Information and Management*, 25:93–102, 1993.

[24] A. Endres. Preface. In *Proceedings of European Symposium on SDE and CASE Technology*, Lecture Notes in Computer Science. Springer-Verlog, June 1991.

[25] A.V. Lamsweerde et al. Generic lifecycle support in the ALMA environment. Tr, Honeywell Systems and Research Center, Minneapolis USA, 1989. Cited From [38].

[26] Excelerator. Reference guide. Index Technology Co., Cambridge, USA, 1987. Cited From [49].

[27] S.I. Feldman. MAKE - a program for maintaining computer programs. *Software Practice Experiences*, 9:255–265, 1979. Standard Reference.

[28] R.I. Ferguson. The beginner's guide to IPSYS TBK. University of Sounderland Occasional Paper 93/3, 1993. `http://osiris.sunderland.ac.uk/rif/metacase/metacase.tools.html`.

[29] P. Findeisen. The Metaview system. University Of Alberta Research Homepage Article, 1994. `http://web.cs.ualberta.ca/~softeng/Metaview/project.html`.

[30] G. Forte. Tools fair: Out of the lab, onto the shelf. *IEEE Software*, pages 70–79, May 1992.

[31] A. Gillies. MetaCASE: One step beyond? In *Proceedings of First International Conference on MetaCASE*. Sunderland, UK, 1995.

[32] A. Isazadeh. Behavioral views for software requirements engineering. Ph.D. Thesis, Queen's University, 1996.

[33] S. Isoda. SoftDA - a computer aided software engineering system. In *Proceedings of Fall Joint Computer Conference*, pages 142–151, 1987. Cited From [34].

[34] S. Isoda, S. Yamamato, H. Kuroki, and A. Oka. Evaluation and introduction of the structured methodology and a CASE tool. *Journal of Systems and Software*, 28:49–58, 1995.

[35] M.A. Jackson. *System Development*. Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[36] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object-Oriented Software Engineering, A Use Case Driven Approach*. Addison Wesley, Reading, Massachusetts, 1992.

[37] M. Jarke. ConceptBase: A deductive object manager for meta databases. Reference Guide, 1996. `http://www-i5.informatik.rwth-aachen.de/CBdoc/cbflyer.html`.

[38] A. Karrer and W. Scacchi. Meta environments for software production. University of Southern California Homepage Article, 1994. `http://www.usc.edu/edu/dept/ATRIUM/index.html`.

[39] S. King, P. Layxell, and S. Williams. CASE 2000: The future of CASE technology. *Software Engineering Journal*, 9:138–139, 1994.

[40] P. Klint. A meta-environment for generating programming environments. *ACM Transactions on Software Engineering and Methodology*, 2:176–201, 1993. Cited From [38].

[41] P. Laamanen. Automation of software product metrics: a proposal for a metamodel based metrics engine. In *Proceedings of First International Conference on MetaCASE*. Sunderland, UK, 1995.

[42] D.A. Lamb. *Software Engineering, Planning for Change*. Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[43] D.A. Lamb, A. Malton, and X. Zhang. Applying the Theory-Model Paradigm. Internal Technical Report 1996IR-01, Queen's University, February 1996.

[44] M. Levene and A. Poulovassilis. The Hypernode model and its associated query language. In *Proceedings of the 5th Jerusalem Conference on Information Technology*, pages 520–530. IEEE Computer Society Press, October 1990.

[45] M. Lioyd. Knowledge based CASE tools: Improving performance using domain specific knowledge. *Software Engineering Journal*, 9, July 1994.

[46] K. Lyytinen and P. Kerola. MetaPHOR: Meta-modeling, principles, hypertext, objects, and repositories. TR 7, Department of Computer Science, University Of Jyväskylä, Finland, December 1994.

[47] G. Maokai and L. Scott. Developing the user interface for the MetaCASE toolset (concept document). JRCASE research report, Macquarie University Joint Research Center for Advanced Systems Engineering, Sydney, Australia, 1996.

[48] P. Martiin. Towards flexible process support with a CASE Shell. In *Proceedings of Advanced Information Systems Engineering CAiSE'94*, Lecture Notes in Computer Science, pages 14–27. Springer-Verlog, June 1994.

[49] P. Martiin, M. Rossi, V. Tahvanainen, and K. Lyytinen. A comparative review of CASE shells: A preliminary framework and research outcomes. *Information and Management*, 25:11–31, 1993.

[50] J. Martin and C. McClure. *Design Techniques for Analysis and Programmers.* Prentice-Hall, Englewood Cliffs, New Jersey, 1985. Cited From [77].

[51] A. McAllister. Modeling concepts for specification environments. Ph.D. Thesis, University of Saskatchewan, 1993. Cited From [29].

[52] C. McClure. *CASE is Software Automation.* Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[53] MetaEdit+. Refernce guide. Metacase Consulting, Jyväskylä, Finland, 1996.

[54] S. Misra. CASE system characteristics: Evaluative framework. *Information and Technology*, 32:415–422, July 1990. Cited From [49].

[55] I. Mitchell and C. Hardy. MetaCASE editorial. In *Proceedings of First International Conference on MetaCASE.* Sunderland, UK, 1995.

[56] R.J. Norman and M. Chen. Working together to integrate CASE. *IEEE Software*, pages 13–16, March 1992.

[57] K.S. Oakes, D. Smith, and E. Morris. Guide to CASE adaption. SEI TR 15, Carnegie-Mellon University, Pittsburgh, USA, 1992.

[58] ObjectMaker. Reference guide. Mark V Systems, Encino, California, 1996. `http://www.markv.com/`.

[59] J.K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, 1 Jacob Way, Reading, Massachusetts, 1994.

[60] D.E. Perry and G.E. Kaiser. Models of software development environments. *IEEE Transactions on Software Engineering*, 17, March 1991.

[61] R.S. Pressman. *Software Engineering, a Practitioners Approach.* McGraw-Hill, 1221 Avenue of Americas, New York, 2 edition, 1987.

[62] QuickSpec. Reference guide. Meta Systems Ltd., Ann Arbor, Michigan, 1989. Cited From [49].

[63] J. Rader, A.W. Brown, and E. Morris. An investigation into the state of the practice of CASE tool integration. SEI TR 15, Carnegie-Mellon University, Pittsburgh, USA, 1993.

[64] Rational. Rational appoints Ivar Jacobson as vice president of business engineering. Company Newspage, 1995. `http://www.rational.com/htdocs/news/pr146.html`.

[65] A. Reeves, M. Marashi, and D. Dudgen. A software design framework or how to support real designers. *Software Engineering Journal*, pages 141–155, July 1995.

[66] T. Rose and M. Jarke. A decision-based configuration process model. In *Proceedings of 12th International Conference on Software Engineering*, pages 316–325. Nice, France, 1990. Cited From [49].

[67] W. Royce. Managing the development of large software systems: Concepts. In *WESCON Proceedings*, August 1970. Cited From [11].

[68] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Deign.* Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[69] A.G. Ryman. The Theory-Model Paradigm in software design. Technical Report 74.048, IBM Co., October 1989.

[70] A.G. Ryman. Foundations of 4thought. In *Proceedings of CAS Conference CASCON'92*, pages 133–155, Toronto, November 1992. IBM CANADA.

[71] A.G. Ryman. Constructing software design theories and models. In *Proceedings of ICSE'93 Workshop: Studies of Software Design*, Lecture Notes in Computer Science, pages 103–114. Springer-Verlog, May 1993.

[72] L. Scott. Hypernode model support for software design CASE. JRCASE Research Report 96/7, Macquarie University Joint Research Center for Advanced Systems Engineering, Sydney, Australia, July 1996.

[73] L. Scott. MetaCASE concept document. JRCASE Research Report 96/8, Macquarie University Joint Research Center for Advanced Systems Engineering, Sydney, Australia, August 1996.

[74] SEEWG. A software engineering environment for the navy. Technical report, NAVMAT Software Engineering Environment Working Group, March 1982. Cited From [11].

[75] S. Shlaer and S. Mellor. *Object Lifecycles Modeling the World in States*. Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

[76] K. Smolander, K. Lyytinen, V. Tahvanainen, and P. Martiin. Metaedit-A flexible graphical environment for methodology modeling. In *Proceedings of Advanced Information Systems Engineering CAiSE'91*, Lecture Notes in Computer Science, pages 168–193. Springer-Verlog, May 1991.

[77] P.G. Sorenson and J.P. Tremblay. Using a Metasystem approach to support and study the design process. In *Proceedings of ICSE'93 Workshop: Studies of Software Design*, Lecture Notes in Computer Science, pages 88–102. Springer-Verlog, May 1993.

[78] P.G. Sorenson, J.P. Tremblay, and A.J. McAllister. The Metaview system for many specification environments. *IEEE Software*, pages 30–38, March 1988.

[79] A.V. Staa and D.D. Cowan. An overview of the Totem software engineering Meta-Environment. Technical Report PUC-Rio.infMCC 35/95, Pontificia Universidade Catolica do Rio de Janeiro, Brazil, November 1995.

[80] S.C. Stobart, J.B. Thompson, and P. Smith. The use, problems, benefits, and future directions of CASE in the UK. *Information and Software Technology*, 33:629–636, 1991. Cited From [31].

[81] Sunderland. Homepage article. University of Sunderland, UK, 1996. `http://osiris.sunderland.ac.uk/rif/metacase/metacase.tools.html`.

[82] W. Teielman and L. Masinter. Interlisp programming environment. *IEEE Computer*, 14:25–33, April 1981. Standard Reference.

[83] I. Thomas. PCTE interfaces: Supporting tools in software engineering environments. *IEEE Software*, 6:15–23, November 1989. Cited From [38].

[84] T.F. Verhoef and A.M.T. Hofstede. Structuring modeling knowledge for CASE Shells. In *Proceedings of Third International Conference on CAiSE*, pages 502–524. Springer-Verlag, May 1991.

[85] I. Vessey, S. L. Jarvenpar, and N. Tractinsky. Evaluation of vendor products: CASE tools as methodology companionships. *Communications of ACM*, 35:90–105, April 1992. Cited From [49].

[86] X. Wang and P. Loucopoulos. The development of Phedias: a CASE Shell. In *Proceedings of CASE'95*, pages 122–131. IEEE Computer Society Press, July 1995.

[87] A.I. Wasserman. Tool integration in software engineering environments. In *Proceedings of Software Engineering Environments*, Lecture Notes in Computer Science, pages 137–149. Springer-Verlog, 1989.

[88] J.L. Whitten, L.D. Bentley, and V.M. Barlow. *Systems Analysis and Design Methods*. Irwin, Burr Ridge, Illinois, 1994.

[89] Y. Yamamoto. An approach to generation of software lifecycle support systems. Ph.D. Thesis, University of Michigan, 1981. Cited From [38].

[90] E. Yourdon. *Modern Structured Analysis.* Yourdon Press, Englewood Cliffs, New Jersey, 1989.

[91] E. Yourdon and L.L. Constantine. *Structured Design.* Yourdon Press, Englewood Cliffs, New Jersey, 1979.

[92] X. Zhang. A Theory-Model formalization of Shlaer-Mellor Object-Oriented Analysis. In *Proceedings of CAS Conference CASCON'94*, pages 324–333, Toronto, November 1994. IBM CANADA.

[93] Y. Zhuang. Object-Oriented modeling in Metaview. MSc. Thesis, University of Alberta, 1994.