Modelling CASE Environments in Systems Development

by

Kalle Lyytinen
Kari Smolander
Veli-Pekka Tahvanainen

University of Jyväskylä
Department of Computer Science
Jyväskylä, Finland

Abstract

Computer Aided Systems Engineering (CASE) environments are pushed into market place daily. Novel features are being added to them, and the area is growing rapidly. However, we lack a more profound understanding of the relationhips between CASE environments and systems development process in general. In this paper we try to make some preliminary steps to bridge this gap. We propose a general framework to analyze and explore systems development process. Using this framework we define the concept of a CASE environment and discuss its role and functions in systems development. Four aspects: organizational, communicational, technical and metalogical are formulated which can be used to identify the effects of CASE upon systems development. The idea of a CASE shell, a design environment for the metalogical aspect and which can be used to "locate" and "program" CASE environments in three other aspects is introduced. We notify 17 dimensions of modifiability in CASE environments and show how these can be used to influence the use of a CASE environments in communicational. organizational and technical aspects. Finally, some research issues raised by our study are discussed.

Keywords: Metamodelling, CASE shells

1. Introduction

Computer Aided Systems Engineering (CASE) is a booming activity in information systems. Numerous CASE products are being marketed and new products are launched nearly daily. And, even more there are programs that are sold as CASE environments, but which hardly can be considered such. The reneissance of CASE has created an expanding and rapidly growing research activity (*cf.* Chikofsky 1988; Penedo et al 1988; Bubenko 1988; Lockemann & Mayr 1986). New research topics are opened just as tool integration in CASE, CASE architectures, database interfaces and techniques associated with CASE, standards, CASE evolution, integration of knowledge representation techniques, or quality assurance methods in CASE. Surprisingly, there is no abundance of research concerning the concept and principles of CASE in relation to systems development in general.

In this paper, we have set out on this topic. The general goal of this paper is to clarify the concept of a CASE environment, and to lay out a framework to study their impacts on systems development. In particular, we want to explore how the modifiability of CASE environments achieved by the exploitation of CASE shells can be used to "reprogram" and "reorganize" system development processes for greater adaptability and productivity. Although most of the present work is concerned with CASE on a very general level, and although we have deliberately avoided making any claims about what might be the scope of CASE, we are nevertheless mostly concerned with so-called upper case *i.e.* CASE in the early stages of systems development.

The paper is organized as follows. First, we present a framework for modelling and metamodelling which has its roots in classic epistemology. Based on this we state some necessary conditions for a CASE environment, and discuss the differences between system development tools and a CASE environment.

In the third section we present three aspects and one "meta-aspect" from which a sounder classification of CASE environments can be derived and which does not apply arbitrary groupings of some technical features found in CASE environments.

The fourth section discusses at some length possible and necessary features of a CASE shell, *i.e.* an environment that can be used to generate customized CASE environments, and thus to "program" [1] the systems development process. We also show how these features are related to the three aspects of CASE environments presented.

Lastly, some research implications are summarized: what should be the proper focus of CASE research in near future, and what problems need to be tackled in order to make most use of CASE environments and in particular of CASE shells.

2. A conceptual framework for systems development

In this section we propose a conceptual framework which helps to illuminate the nature of systems development process. Basic concepts that clarify various elements and functions in systems development process are exposed. Then, we will use the framework to shed light on the role of a CASE environment in a systems development process and suggest two criteria to define the concept CASE more accurately.

2.1. A model of Systems Development

Generally speaking, systems development can be defined as an *inquiry and change process that concerns a specific field of action called here a field of phenomena* (Lyytinen 1987). This inquiry and change process is social in nature, *i.e.* it involves more than two people who may take part in this process in different *roles*. In order to explore the nature of this process we must distinguish three fundamental domains of systems development: (1) *the real world, i.e.* the basic "stuff" of inquiry and change: organizations, data, communications,

[1] The word "program" does not mean that we would prefer a rigid approach to system development, where every step is exactly prescribed and that remains unchanged from one development situation to another. However, we do assume here, that all systems work exhibits some systematical pattern, although the particular tasks and their sequence may change from time to time.

activities, computers, files and the like; (2) *conceptualizations of the real world, i.e.* organizational theories, theories of data and its modelling, or concepts used to understand and interpret the "reality" , and (3) *descriptions of these conceptualizations, i.e.* representations of organizational activities, data flows, data structures, algorithms etc.

In more detail we should distinguish between (figure 1):

- *a field of phenomena, i.e.* a set of states of things in the real world, "reality".
- *a conceptual structure,* a model in the beholder's mind, which conditions and affects how the field of phenomena is conceived by him as certain objects, events and so on.
- *a level of abstraction,* a level of granularity related to the conceptual structure. The level of abstraction determines which sorts of objects are constituted in the field of phenomena, and which parts therein are considered to be relevant in different stages of systems development.
- *a target system,* which is formed from a part of the field of phenomena when seen through a particular conceptual structure on a particular level of abstraction.
- *a description language,* a notation, in which the the target system is represented and communicated.

The framework thus distinguishes between the "reality" of systems development (that is or will be analyzed and changed), the developers' concepts and theories of it (conceptual structure), their "matching" as a particular "target system", and various representations made out of the target system in description languages. Consider an inventory system. The field of phenomena consists of all possible things that can happen and take

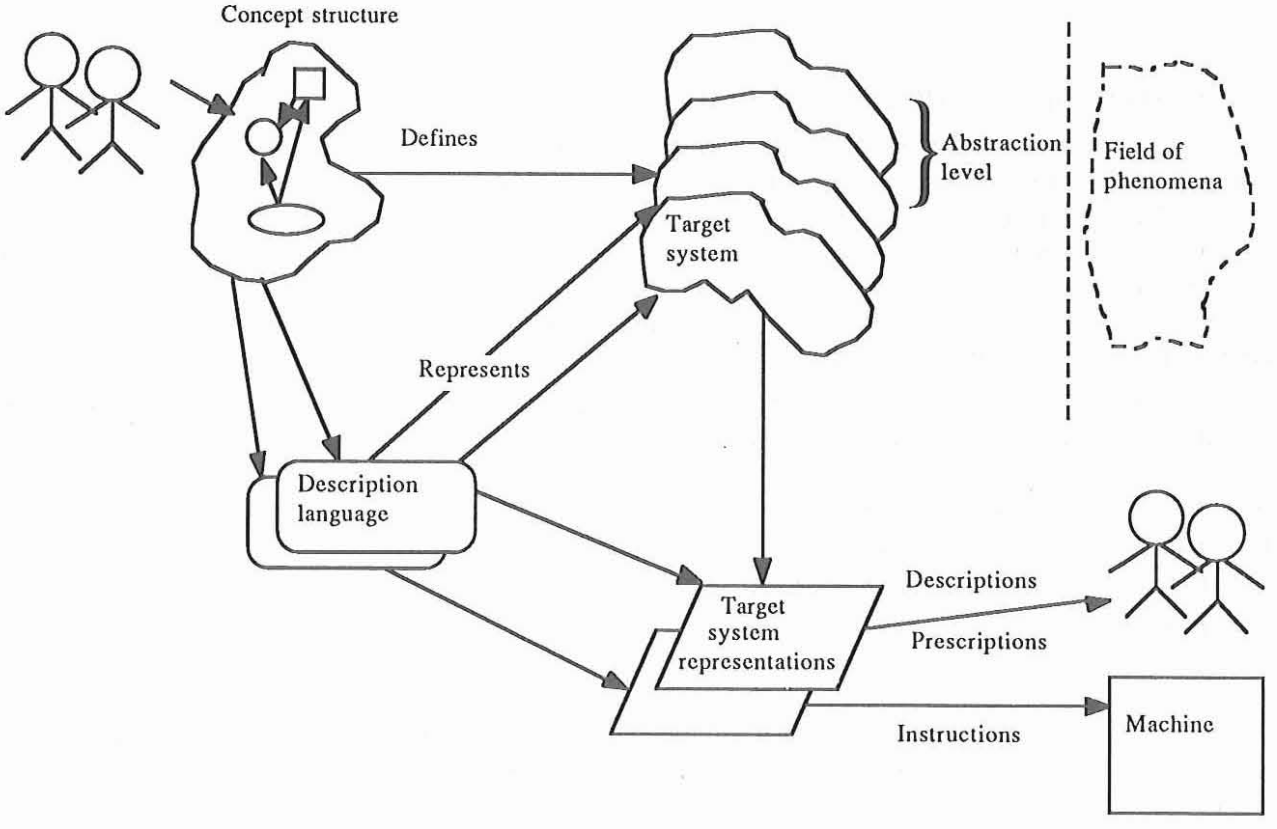

Figure 1. A Model of Systems Development

place in the inventory system. The conceptual structure is formed by the concepts and theories the developers (systems analysts) have about the role of information systems in the inventory systems and all concepts that help to understand, explain and predict the behavior of the system. The conceptual structure is thus partly constituted by "methods" and "approaches" of systems development learned and used by systems analysts, partly by the knowledge they have about inventory systems and their role in organizational action. The target system is a particular "model" of an inventory system derived by the analyst when the conceptual structure is "matched" with the inventory control problem in a particular setting. Description languages are notations that the analysts use to represent the target system for analysis, communication and review such as a data flow model of the current inventory system.

Although these elements are represented here as independent factors of a development situation, they in fact are not independent, and therefore each one of them must be understood only in relation to the other elements. The distinctions are thus not absolute, and they need to be made only when necessary. For example, the conceptual structure and the level of abstraction are mutually dependent, but neither of them can be explained by and reduced to the other. One and the same conceptual structure can convey several levels of abstraction, and one and the same level of abstraction can be found in several different conceptual structures. For example, some data models can include conceptual structures of data on several levels of abstraction (storage dependent, and independent concepts).

In the same vein, representations of target systems can be read on several levels of abstraction depending on which conceptual structure is being applied. For example, a piece of program code can be read as a representation of semiotic relationships (salary-field denotes to real salaries), as a representation of a formal expression (salary-field has five integers and two decimals), or as a model of computer memory assignment (a salary-field takes one word of the main memory). Finally, the same conceptual structure can be dressed into different description languages *i.e.* the same target system can have several semantically equivalent representations (see for example Venable & Truex 1988).

2.2. Description languages in system development

The main focus and interest in systems development lies naturally in description languages as they make the conceptualizations of the field of phenomena shareable among the members of the development group. In other words, only through description languages systems development as social activity is possible. This obvious conclusion becomes clear if we look at any book or research article on systems development. A majority of the literature discusses new and better ways to describe the target systems. The other components of our framework help to understand that the descriptions are *of* some field of phenomena and that they are *based on* some more or less explicit conceptualizations, "theories" , of the field of phenomena.

The representations of target systems in description languages serve several important functions in systems development. First they are a *means of communication.* Second, they are a means of *analysis, understanding and prediction* of the structure and behavior of the target systems. In other words, descriptions are instrumental in inquiring the field of phenomena and obtaining knowledge of it and communicating this knowledge to other participants during the development process. Therefore, it is necessary to consider in more detail the types of communications and types of analysis that take place in systems development.

First, the communications can have either a *descriptive* or *prescriptive* purpose. In brief, the representations of target systems can either describe states of things as they are here and now (or have been in the past), or they can prescribe states of things that are intended to take place in the future. (One might suggest an intermediate state, a description of a system to come. This could however be counted as a special case of description.) An example of representations having descriptive purpose is a representation of data flows in the current system. An example of prescriptive representation is the logical data flow model of the future system.

Another dimension in our analysis of communications is whether the communication takes place *between people* or *between people and computers.*

In the former case the communication either clarifies, or explains the structure and behavior of the target system. Sometimes the communications can also prescribe behaviour of the people taking part in the target system (user documentation). In the latter case the communication is not communication in a true sense, it is

rather a way of instructing the behaviors to the computer system (*i.e.* programming in the real machine-oriented meaning).

In the *inquiry* mode (as opposed to the communication mode discussed above) the representations serve as a means to acquire, analyze and synthetize knowledge of a field of phenomena. This means that the representations enable checking the validity of the knowledge obtained. Several levels of correctness can be distinguished: the syntactic correctness, semantic consistency of the representations, their correspondence with developer's intentions and desires, or their pragmatic usefulness in proposing changes into the field of phenomena.

In practice, the representations of a target system can be used in several types of communications and inquiry at the same time. For example, a piece of program source code may be considered

(1) to serve descriptive communications, as it describes the functioning of the computer as well as the information system at a (quite low) level of abstraction.

(2) to serve prescriptive communications, in stating *how* the computer should work;

(3) it may also be called an *instructive* description from the human programmer to the computer;

(4) it also serves to communicate information between people about the functioning of the system; and

(5) finally, it may also be an object to study the syntactic and semantic correctness of the program (program verification) or to higlight the usefulness of the program to its users (although rarely).

2.3. The nature of system development process

Using this framework as a starting point we can now interpret the design process as an inquiry and change process that involves matching of conceptual structures with the field of phenomena for the purpose of description and prescription (concerning both human beings and computers). It is an inquiry to the extent it involves the matching of the conceptual structures (interpreting a set of observations through a theory) with a field of phenomena to yield a set of target systems and their representations in a host of description languages that can be communicated, analyzed, and manipulated by a group of people involved in the development process. It is a change process to the extent it involves the matching of a conceptual structure with a field of phenomena to generate a new field of phenomena and communicating, analyzing and manipulating these phenomena through prescriptive representations. Usually the description (or a systematic pattern followed by developers) of how the matching takes place and how the descriptions are derived is called a *method.*[2] When a method is supported by some instrument (a template, a questionnaire, or a computer program) this is called a development *tool.* [3]

Representations of target systems in systems development (both in the descriptive and prescriptive sense) can be an object of a high variety of activities. Representations can be:

- *produced*, automatically or by hand,
- *transformed*, e.g. from a level of abstraction to another (by people or by computer),
- *maintained*, *i.e.* when minor changes are made to them by people,
- *approved* or *inspected* by a group of people,
- *managed*; *i.e.* named, ordered, accessed, secured, or subjected to version control,
- *translated*, e.g. from one description language to another.

A systems development process forms thus a sequence of (partly) parallel activities where each activity takes some representations of target systems as an input and produces new representations of target systems as an

---

[2] We do not want to make here a distinction between a method and a technique as is sometimes done. In our opinion, the concepts of "method" , "technique" and "methodology" are so hopelessly blurred that clear distinctions between them are very difficult to make and to discuss them here any further would be beside the point.

[3] For example, structured analysis involves a conceptual structure (consisting of concepts like data flow, store etc.), a description language (graphical symbols and syntax) that are "matched" with some application domain (a field of phenomena). The method of SA involves in what sequence the representations are derived (current physical system model, logical model etc). The tools of SA would include templates, an editor to write down and edit texts (minispecs), or a CASE tool to draw SA diagrams and to store them.

output.

Improvements in systems development can be achieved in at least three ways:

(1) by deciding on which target systems are selected (for description and prescription) and how they are represented (a question of methods),

(2) by affecting in what sequence and in what organizational form different sorts of activities (*i.e.* taking different representations as input and producing different representations as outputs) should be carried out (the question of phase models and project organization),

(3) and by adopting new facilities into use to carry out such activities (the question of tools).

As these improvements are highly interdependent, a more encompassing notion of *systems development methodology* (SDM) has emerged. Though no satisfactory and wholly accepted notion of systems development methodology exist, we can define a systems development methodology as an *organized collection of conceptual structures, description languages, activities, prescriptions for patterns of activities, organizational forms, and finally facilities that help to carry out activities.* The purpose of a SDM is to help the development group to inquire and change some field of phenomena, that is to perceive, generate, assess, control and carry out change actions in a set of target systems.

Using the framework just presented we can more easily grasp the concept of a CASE environment. In brief, a CASE environment "implements" a specific SDM in a computer supported and readable form. A CASE environment is a generic facility that supports several activities that take target system representations as inputs and produces other types of target system reprensentations as outputs according to a SDM.[4] The phrase "*implements*" a specific SDM is here essential. It implies that not any type of computer implemented activity that deals with target system descriptions forms a CASE environment (consider using a text-processor). More specific requirements can be derived from the phrase "implements a SDM" This will be discussed next.

3. Some steps towards a more precise definition of CASE

Formal definitions of a CASE concept abound in the literature (Chikofsky 1988; Bubenko 1988; Penedo et al 1988). We are not striving here for a formal definition. Instead, our model of systems development helps to suggest some clear criteria which a CASE environment must satisfy. In other words we will provide two necessary and sufficient features, for a "true" CASE system.

First, a CASE environment must implement several conceptual structures.[5] This requirement is necessary, if the environment is to support any methodology (SDM) at all, because all methodologies contain in a more or less explicit form a multitude of conceptual structures to describe, interpret and prescribe a field of phenomena.[6] This means that the CASE environment should encapsulate some "knowledge" of how description languages are used and how to derive target system representations in them and what criteria determine their validity. This requirement distinguishes CASE environments from drawing programs or text formatters that do not have any conceptual structure behind their use *i.e.* they just help to draw some figures which can

---

[4] Note that this is a very broad "definition" and could include for example reverse engineering and automatic programming.

[5] We think that this criterion to support several conceptual structures distinguishes CASE environments from CASE tools that provide often quite comprehensive support for a particular method.

[6] There are several different ways how conceptual structures in a SDM can be connected. The most usual situation is, that no conceptual structure is disjoint from the other conceptual structures *i.e.* that each conceptual structure has at least one common concept with some other conceptual structure. A more stringent requirement is that all conceptual structures have at least one common concept. Usually this situation prevails with conceptual structures on the same level of abstraction (*cf.* different steps in SA, which all use the concept of process). In some other occasion conceptual structures on different levels of abstraction can have one or a set of common concepts by which they are connected. The concept of concept sharing corresponds in systems development process a subsetting of target system representations by similarity. Another way to connect conceptual structures is to provide mapping mechanisms by which a concept is mapped onto a set (or a powerset of) conceptual structures (*cf.* a mapping of an ERA-schema into a relational schema). The concept of concept mapping corresponds in a systems development process a transformation by some tool from one target system reresentation to another target system representation.

not be further interpreted by programs. When this criterion is used it is also questionable, whether a 4GL can be considered a CASE environment, because their primary focus is instructing (programming) the computer (though on a quite high level). A stronger requirement is, whether the CASE environment can be customized to new and different conceptual structures, or variations of one conceptual structure. This makes CASE environments modifiable, a feature of CASE shells, as we shall show in section 5.

Second, a CASE environment should embrace several levels of abstraction and transitions between them. Thus, if the CASE environment implements conceptual structures and description languages that all belong to one level of abstraction, it is not a true CASE environment. In this situation, the environment does not support the inquiry and the change process as it moves from one abstraction level to another e.g. from more organization oriented conceptual structures to more machine oriented ones. This becomes perhaps more understandable, if one thinks that a level of abstraction often coincides with a definite phase of the systems development process. So, for example, target system descriptions during the requirement specification phase and the source code (Dart et al. 1987) (descriptions of computer behaviors) both describe the system at distinct levels of abstraction. This requirements threatens the status of integrated programming support environments (IPSE) *qua* CASE environments, because they usually help to produce and maintain the source code. However, if an IPSE does produce or make use of higher-level representations and provides functions for their maintenance it should be considered a "true" CASE environment.

All in all, to reiterate the two requirements for a CASE environment are:

(1) it has to embed several conceptual structures and descriptions languages embedded in a SDM, and

(2) it has to support several levels of abstraction at which the development process takes place and support activities that concern target system representations on that level.

Clearly, these two criteria form a minimum, and more restrictive conditions can be proposed. However, even with these two conditions several tools marketed as CASE environments can be left outside the field of CASE.

More restrictive criteria that can be considered are the following:

(1) Does the environment support only production and maintenance of target system representations that describe or prescribe for human beings *i.e.* is it a mere analysis and design tool, or does it also generate instructions or parts of instructions to the computer *i.e.* does it involve back-end functions?

(2) Does the environment implement and maintain a model of a necessary pattern of activities to be carried out during the development process, *i.e.* does it have a representation of actitivities and their relationships (phase model) that can be used to guide and monitor the development process?

(3) Does the environment implement and maintain a model of the organizational form of the development process *i.e.* does it model a set of roles and their relationships to activities so that it can be used to guide, coordinate and monitor various activities during the development process?

(4) What sorts of activities related to target system representations does the environment support in addition to producing and maintaining them? Does the environment support tranformation, translation, and review activities? How well can it support the management of various versions of representations, define access controls, and order them in different ways?

(5) Does the environment make use of existing descriptions and implementations? Does it have e.g. reverse engineering or code reuse facilities? (Of course, not only program code but various other forms of descriptions can be reused. (E.g. cf. Madhavi et al., 1985.))

Each of these defines a new aspect that must be considered if a CASE environment is to support wholly a specific SDM.

4. How to classify CASE tools

There are several perspectives which can be used to analyze CASE environments (in other words there are alternative conceptual structures to interpret the field of CASE). These perspectives suggest alternative ways to classify CASE environments as they select a different set of properties which can be used as a basis for

classification. For example, one can choose a single property or a set of technical features in a CASE environment. A single property could be the type of description language implemented by the CASE environment (graphical/linear). A set of technical features could be operating systems where it can be run, type of the DBMS embedded in the CASE environment, programming language interfaces, 4 GL interfaces, data dictionaries supported etc. Although these classifications are useful (e.g. Dart et al. 1987), especially when acquiring a CASE environment, their problem is that they lack a more profound theoretical foundation and produce quite haphazard classifications. Moreover, a great number of single properties and features can be enumerated leading to a cumbersome and unstructured taxonomy which often obscures the essential differences between various CASE environments.

We believe that a more fruitful way is to view CASE environments as to form an integral part within a larger information systems development environment. In this approach a CASE environment is not conceived solely as a technical object with clearly identifiable technical features. Instead, we focus on the functions and role of a CASE environment in systems development, *i.e.* how it helps to represent target systems and to communicate these representations to various stakeholders during the development process and how it helps to carry out various inquiring activities. The "help" implies here that the environment makes system development activities easier to carry out and/or that their outcomes have higher quality (less errors, more efficient prescriptions, easier to modify the descriptions etc). In addition, a CASE environment can be used to monitor and manage the development activities and to keep track of the consumption of scarce organizational resources.

Our framework of systems development illustrates that the classification of CASE environments is a multidimensional problem. It is not possible to divide CASE environments into some distinct (disjoint and exhaustive) classes and say, for example, that there are five kinds of CASE environments. Instead, we are obliged to think of a rich variety of aspects of information systems development and how they relate to the use of CASE environments. In the following, four aspects are defined which we believe, are essential in classifying the use of CASE environments in different systems development situations. These are: a communicational, an organizational, a technical, and a metalogical aspect.

4.1. The communicational aspect

In the communicational aspect the central issue is *the exchange of target system representations among the developers* (including end-users' representatives). The communicational aspect focuses on the type and intensity of communications during systems development *i.e.* how and what types of target system representations are shared by the members of the development group (or others concerned). The major purpose in the communicational aspect is to understand how the developers can attain a common understanding of the relevant target systems and their behaviors. To analyze the communicational aspect in more depth three subdimensions can be applied (fig. 2). These are: supported activities, type of users, and mode of communication. Each particular CASE environment can be situated somewhere in the space formed by these three dimensions.

Supported activities are those systems development activities which are supported by a CASE environment. The activities subdimension covers the following issues:

- which activities are supported: creation, transformation, management, review, or translation of representations; and
- in which phase (activity sequence) of the systems development process.

The *type of users* subdimension delineates two issues. First, a CASE environment can support communications between several users simultaenously, or it can be used only by one user. In the latter situation the environment can support merely inquiry that relates to an individual analyst's or end-user's activity. Second, a CASE environment can distinguish between several types of users *i.e.* different roles adopted during the development process. Typical roles are a manager, a project leader, an analyst, and an end-user.[7] If user-to-user communications are allowed they can take place within users acting in the same role or between users having a different role.

[7] We use here the term "end-user" to differentiate between CASE users and users of the information system (end-users).

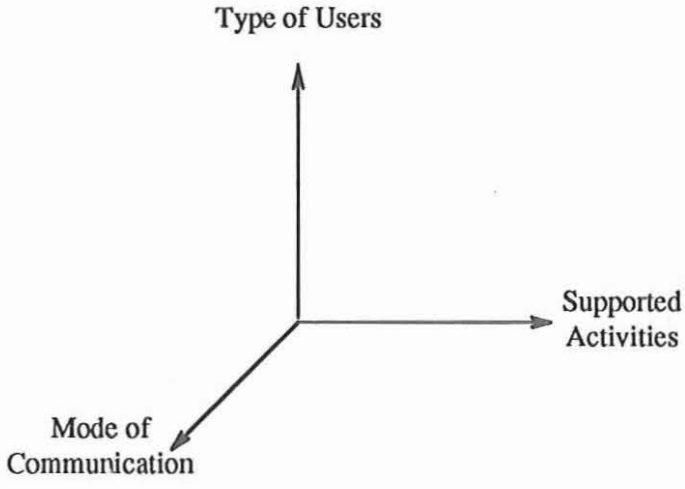

Figure 2. Dimensions of the Communicational Aspect

The *mode of communication* subdimension illustrates communication forms and structure. The communications can in one extreme support several forms of communication ranging from visual forms (pictures etc.) and textual communication upto oral communications. In the other extreme the communication form is fixed only to one (usually textual communication). Another aspect is that communication can be highly structured or it can have a more free-form structure. An example of highly-structured communications are communications through a common description (data) base where the communication takes place through updating and querying a shared description base. An example of free-form communication is the electronic mail. Semi-formal communications that are situated in-between these two extremes are also possible (such as prestructured protocols for a set of dialogues).

4.2. The organizational aspect

In the organizational aspect, the central focus is in *control.* The aspect delineates those qualities of CASE environments by which it is possible to manage and control systems development process. Since the use of a CASE environment always takes place in an organization, the nature of the CASE environment (whether it is intended for one or multiple users) has little influence on this aspect. Three subdimensions under the organizational aspect can be noted (figure 3).

The first dimension, *control/performance,* defines the primary organizational strategy to use a CASE environment: to which amount it is used to control the development process, and to which extent it is applied to carry out the actual systems development activities. This distinction is clearly visible for example in tools that help to carry out analysis and design tasks, and those that are used for project management purposes.

The second dimension, *compelling/volitional,* tells how a CASE environment guides analysts' or other developers' work. A CASE environment can force a developer to act strictly in a predefined way or it can give him more freedom so that he or she can use methods being supported in ways he or she sees as best fitting to the situation. For example, some environments state a strict sequence of steps in which the target system representation is to be derived (consider some SA-based tools), or the representation order is totally free (e.g. PSL/PSA).

The third dimension, *hierarchical/lateral,* defines how the use of a CASE environment is organized. If the organizational environment is hierarchical, a set of roles is assumed which have clearly defined command, authority, and reporting relationships. In this situation the communication flows either upwards or downwards in a hierarchy. Typical examples of hierarchical communication models are project management models, which can state several hierarchical layers of control. In a lateral organizational situation the flow of communication is horizontal, and often volitional and it does not presuppose a hierarchical command structure.

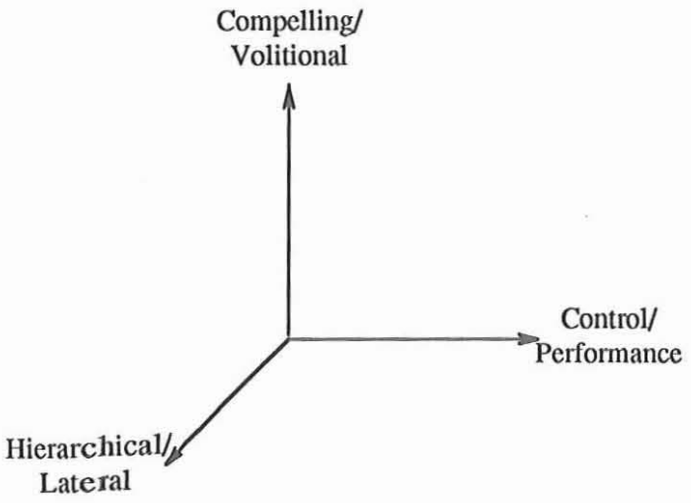

Figure 3. Dimensions of the Organizational Aspect

4.3. The technical aspect

The technical aspect suggests in all likelihood the most straightforward and common way to classify CASE environments. This aspect focuses on features that affect *how systems development is carried out and what activities must necessarily take place.* The subdimensions under the technical aspect are the following: design principles, the level of abstraction, the application area, and the interface (fig. 4).

The first dimension, the *design principles* focuses on generic stucturing mechanisms to relate conceptual structures in different methods together. Usually each CASE environment applies some generic design principle to connect different target system representations and to ease the move from one to another. Examples of typical design principles are data-centered or process-centered design principles. Some CASE environments can even support several desing principles simultaneously, or mix them.

The level of abstraction concerns those abstraction levels that are being supported in the CASE environment. The scope of abstraction in a CASE environment may range from overall design (a high-level abstraction) to machine-oriented design (a low-level abstraction). Several levels are supposed to be covered by a CASE environment (*cf.* Lockemann & Mayr 1986).

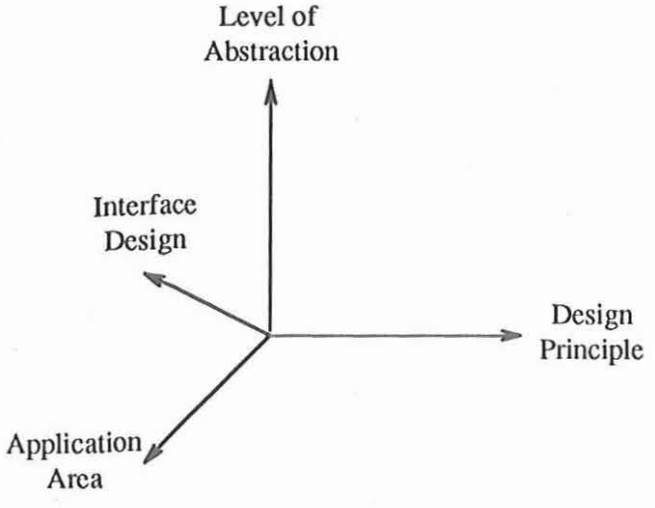

Figure 4. Dimensions of the Technical Aspect

The application area discloses the types of target systems for which the CASE environment can be used. Examples of application areas are embedded systems, office systems, expert systems etc. Each application usually requires some specialized conceptual structures to be adopted that help inquiry and problem-solving acitivities in that specific field of phenomena. Examples of such specific conceptual structures are dynamic aspects, concurrency and communication mechanisms in embedded systems, different types of data, exceptions, and distributed nature of applications in office systems and so forth.

The *interface design* determines the visible features of the CASE environment. The subdimensions of the interface design are: interaction mode (interactive, batch), interaction style (command language, menu-driven, icon-based), and functionality (facilities for query and report generation, programming tools) (*cf.* Dart et al. 1987).

It is clear that the technical aspects of a CASE environment are closely related to those conceptual structures and description languages that have been chosen as a basis to develop the CASE environment. This leads us to the next aspect which is mainly interested in how CASE environments and conceptual structures can be fitted together *i.e.* the metalogical aspect.

4.4. The metalogical aspect

One can consider the metalogical aspect as a meta-aspect for all the previous aspects, *i.e.* those aspects that affect how the CASE environment is used in system development situations, which activities are being supported, how the development group is organized and so forth. The metalogical aspect thus concerns the ''second order'' analysis, design and change of system development situations by ''programming'' the CASE environment. The metalogical aspect focuses on how a CASE environment and a chosen particular conceptual structure can be made logically consistent so that the CASE environment can serve to develop target system representations, analyze and communicate them in some preconceived manner that corresponds to the chosen conceptual structure. In general, the metalogical aspect defines how easily a CASE environment can be customized along the three aspects (and their subdimensions) so that it can better satisfy the developer's desires and needs. If a CASE environment can be extensively customized in all three aspects the developers can by themselves locate the CASE environment in all three other dimensions. If the modification is possible in several aspects we can talk about *CASE shells i.e.* environments that help to customize CASE environments to support an arbitrary methodology or to add a new method to the existing collection of methods. Usually these methods have some common parts in their conceptual structure. A general research question in the metalogical aspect is what levels of customization are possible and what customization strategies are useful in different situations? This points out the need to consider in more detail the concept of CASE shells and different levels of customization achieved by them. This will be clarified in the next section.

5. CASE shell environments

A key issue in the metalogical aspect is the ability to create *CASE shells* [8] *i.e.* programs, that generate CASE environments with specific features. We shall now examine the concept of CASE shells more closely and take a take fresh look how they are relate to the framework we have presented above.

Most currently marketed CASE environments support only a rigid set of conceptual structures and therefore do not qualify as CASE shells. However, it is quite common that they offer some functionality that is typical for CASE shells. One typical feature offered is that the notation used to represent the chosen target system is customizable, so that different symbols are permitted (''boxological sugar''). In this case the customization concerns only the technical aspect, and therein only the interface design (interaction form). To distinguish CASE shells more clearly from CASE environments a more thorough analysis is needed.

[8] The term 'CASE shell' was coined by Bubenko (1988), who intended it to be used in a similar manner as when talking about an expert system shell. Therefore, the term 'shell' here should not be mixed with e.g. an alternative command interpreter. Possible synonyms would be a 'CASE environment generator' or a 'Methodology Engineering Environment' (cf. Kumar & Welke, 1988).

5.1. Three dimensions of the concept CASE shell

In principle, we can distinguish three components of a CASE environment that can be changed. These three components are: (1) *the linguistic component,* (2) *the functions component,* and (3) *the mechanism component* of the CASE environment. The first component defines the concept structures, and the description languages deployed and supported by the environment. The second component determines the types of functions and activities that can be accomplished within the CASE environment. The third component determines the principal underlying mechanisms that provide the visible interface and functionality of the environment. In each component there are several subcomponents that can be modified and which determine how powerful the CASE shell is.

5.1.1. The Linguistic Component of a CASE shell

In general, the linguistic component of a CASE shell defines the scope, depth and variability of the target system representations (description languages) that can be produced, maintained and communicated within the CASE environment. The desciption languages embedded into the CASE environment can be modified on several consecutive levels of modifiability:

- **notational modifiability facility** defines to what extent the symbols can be modified in the description language (usage of nick-names, aliases or different graphical elements);
- **a notational variation facility** defines to what extent the environment allows for notational variation between graphical and lexical languages and supports them both simultaneously;
- **a notational translation facility** defines to what extent the environment can translate from one notational variation to another. Two principal situations can be distinguished: from graphical to lexical notation (the easier one), and from lexical to graphical notation (the difficult one);
- **a syntactic extension facility** defines to what extent the environment supports extending the description language with new syntactic elements (new object, and attribute types);
- **a syntactic definition facility** defines to what extent the environment can support defining a new description language into the environment (including rules for semantic consistency checking etc.);
- **a syntactic translation facility** defines to what extent the environment can support in specifying a translation mechanism from one description language to another (including heuristics for solution generation, consistency checking etc.);
- **a type specification facility** defines to what extent the environment can support in specifying new data types such as figures, voice data etc.

In a "conventional" CASE environment the conceptual structures are fixed and therefore they can at most support syntactic extension and partial notational variation. It may, of course, allow some variation in notations. Obviously, a CASE environment that also serves as a CASE shell must permit to change its underlying conceptual structures. There are different approaches to achieve this. Either, the shell does not convey any fixed conceptual structures, but instead provides only a set of primitive tools from which more elaborated conceptual structures can be built (*i.e.* a set of conceptual constructor operators). In this situation the derived conceptual structures and description languages in the CASE environment are unrelated to the primitives found on the CASE shell level. Or, there is some basic set of conceptual structures, from which the various concept structures are derived (*i.e.* the conceptual structure being "transformed" is mapped onto conceptual structures provided by the CASE shell). In this case the conceptual structures on the level of CASE shell and on the level of CASE environment are related by subsetting and mapping operators. An example of this approach is SEM that is based on PSL (Teichroew et al. 1980). If several methods are to be combined, the latter approach may be preferable, because there is a common basis for all the different description languages. However, if maximal flexibility is demanded, a shell of the former kind is likely to be more useful.

5.1.2. The functions of a CASE shell

In general, the functions component of a CASE shell defines the scope of computer supported activities dealing with target system representations and the way how they can be modified. The support rendered by functions component can be extended to cover nearly all activities including automatic production of target system

representations, transformations, maintenance and review, and management of descriptions.

The functions component of a CASE shell must therefore render several types of modifications into functionality of the CASE environment. At least the following types of modifications can be distinguished:

- **a query definition facility** determines to what extent the environment enables extracting descriptions from a description base under different conditions and formats;
- **a report definition facility** defines to what extent the environment supports specification and execution of report specifications that are derived from the description base;
- **an inference definition facility** defines to what extent and what types of inferences the environments enables to make on extracted descriptions;
- **a storage definition facility** defines to what extent and what types of descriptions (type, format) can be stored in the description base and how their connections can be implemented and managed;
- **a usage definition facility** defines to what extent and what types of uses of the environment can be described, supported and coordinated by the environment.

The first four facilities together are instrumental in modifying support provided by the CASE environment in such tasks as consistency checking, verifying correctness of descriptions, or design aid. They are also needed in specifying customized documentation for different groups of IS users. Finally, they are necessary, if the environment is to enforce systematic application of methods *i.e.* to derive target system representations in a step-by-step manner, or to transform target system representations from one representation to another using heuristic or algorithmic techniques. The inference definition facility may also be needed in defining and supporting ways to use the descriptions to derive solutions to encountered design problems (use of heuristics and knowledge representation schemes).

Storage definition facilities ease modification of such important functions of a CASE environment as version control and management, requirements tracking mechanisms, or specification of responsible persons to maintain and approve target system reprentations (review and control management).

Usage definition facility can be used to configure such features of the CASE environment as access rights, access control or guidance mechanisms to use the environment in different situations. The usage definition facility determines also the style and mode of interaction.

5.1.3. The mechanism component of a CASE shell

The mechanism component of a CASE shell is needed to configure the CASE environment using different systems components including the underlying DBMS, target DBMSs, interfaced data dictionaries, 4GL's, communication protocols etc. The scope of modifiability in this component defines the openness and portability of the CASE environment. Thus, the mechanism component defines the internal and external interfaces by which the functions and support of the environment are provided. Several types of mechanisms can be distinguished:

- **Export/Import definition mechanisms** define the extent and ease by which target system representations can be exported from the environment or imported into the environment. In other words, to what extent can the environment be easily interfaced with other components of an IS resource management environment. Typical components of such an environment are: data dictionaries (e.g. IRDS), data base management systems (e.g. SQL), other CASE environments, text processing systems (ODA, DCA, Ventura), application generators, or source code control systems.
- **Data Management Definition Mechanisms** define the extent and ease by which the target systems representations can be stored and managed in different data base management systems *i.e.* the portability of the CASE environment over different DBMS.
- **Interface Definition Mechanisms** define the extent and ease by which the interface design of the environment can be adopted into different interface standards (Presentation Manager, X-Windows, terminal compatibility);
- **Data Communication Definition Mechanisms** define the extent to which the environment supports various data communication protocols (TCP/IP, OSI, X.400) etc.

- **Operating System Interface Mechanisms** define the extent to which the environment can be run under different operating systems (OS/2, UNIX).

5.2. The metalogical Design of CASE environments

The distinguished three components of a CASE shell make possible a metalogical design of a rich variety of CASE environments. Depending on the degree of modifiability provided by the CASE shell, different aspects of CASE environments can be changed, and "programmed" To obtain a more refined picture of the depth and scope of the "programmability" of CASE environments we shall analyze in more detail the degree of modifiability of CASE shells. Results of this analysis are depicted in tables 1, 2, and 3. The tables show how a subcomponent of a CASE shell (rows) can be used to "design" different aspects of a CASE environment and its use (columns). Table 1 illustrates the relationship for the linguistic component, table 2 for the functions component, and table 3 for the mechanisms component, respectively.

When we study the tables we can note the following. The level of modifiability provided by existing CASE environments such as notational variation, and notational modifiability are not powerful enough to allow for real "design" of the CASE environment. The possibility to extend the description languages used, and especially facilities to define new languages, and to specify translations from one language to another are here essential. Only these components can help to generate a sufficiently flexible environment, *i.e.* to change design principles, to cover several (or new) levels of abstraction, to adopt the environment to new application areas, and thereby also to affect the interface of the CASE environment; all requirements that are needed to have a complete development environment modifiability (*cf.* Sorenson et al. 1988).

When we study the functions component, we can see that the modifiability of the structure component must be supplemented with a powerful and flexible CASE "programming" environment. This environment must offer query and report specification facilities. Moreover, the inference facility must implement in addition to common arithmetic operators also logical inference and other inference mechanisms (rule-based environments)

Aspect	Communicational			Organizational			Technical			
Language	Supported Activities	Type of Users	Mode of Communication	Control/ Performance	Compelling/ Volitional	Hierarchical/ Lateral	Design Principle	Level of Abstraction	Application Area	Interface Design
Notational Modifiability	—	—	+	—	—	—	—	—	+	+
Notational Variation	+	—	+	—	—	—	—	—	+	+
Notational Translation	+	—	+	—	—	—	—	—	+	+
Syntactic Extension	+	+	+	+	—	+	—	—	+	+
Syntactic Definition	+	+	+	+	—	+	+	+	+	+
Syntactic Translation	+	+	+	+	—	+	+	+	+	+
Type Specification	+	—	+	—	—	—	—	—	+	+

Table 1. CASE environment modifiability by language

Aspect	Communicational			Organizational			Technical			
Functions	Supported Activities	Type of Users	Mode of Communication	Control/ Performance	Compelling/ Volitional	Hierarchical/ Lateral	Design Principle	Level of Abstraction	Application Area	Interface Design
Query Definition	+	—	+	+	+	—	+	—	+	+
Report Definition	+	—	+	+	+	+	+	—	+	+
Inference Definition	+	+	—	+	+	—	+	+	+	+
Storage Definition	+	—	+	+	—	+	+	+	—	—
Usage Definition	+	+	+	+	+	+	+	—	+	+

Table 2. CASE environment modifiability by functions

employed in knowledge based systems. Only in this way can the scope of supported activities be extended to include new organizational aspects, and to encapsulate technical aspects such as more effective design principles for adoption to different application areas. Finally, only in this way can the interface design be flexibly configured and designed.

The mechanism aspect is obviously the least important in generating a rich variety of different types of CASE environments. Its key purpose is to serve as a platform for portability and openness (which are important technical and economic concerns). For example the data communication protocols supported by the CASE environment can be decisive in defining the scope and type of communications that can take place through the CASE environment.

6. Research Implications

The framework presented above poses several interesting research questions. Here we shall discuss some of them in a preliminary fashion. A more detailed analysis of the research implications must be done in papers to come. The following two research issues will be treated: the current focus in the CASE research, and the role of CASE shells in the CASE research.

6.1. The focus of the CASE research

In our opinion, current CASE research is often too narrowly focused. Many of the current research projects concentrate on a single technical feature of the CASE environments such as a desirable architecture, user interface, application of knowledge representation techniques, object-based management techniques and so forth. What we lack is a thorough understanding of how these well motivated technical advances can improve the state of art in systems development. To achieve this we need more theoretical and empirical studies on the nature of systems development and how CASE environments can affect it. In this paper we have made some initial steps in this direction by showing that CASE environments are to be looked upon from a wider perspective which includes:

- **Organizational aspects** *i.e.* organizational forms recognized and supported by the CASE environment;

Aspect	Communicational			Organizational			Technical			
Mechanism	Supported Activities	Type of Users	Mode of Communication	Control/ Performance	Compelling/ Volitional	Hierarchical/ Lateral	Design Principle	Level of Abstraction	Application Area	Interface Design
Export/ Import	+	+	—	+	+	+	—	+	+	—
DM Definition	—	—	—	—	—	—	—	—	+	—
Interface Definition	+	+	+	—	—	—	—	—	—	+
Data Communication	+	+	+	+	—	+	—	—	—	+
Operating System	—	—	—	—	—	—	—	—	+	—

Table 3. CASE environment modifiability by mechanism

- **Communicational aspects** *i.e.* type and intensity of interactions that are needed in systems development among different developers and how these are changed and supported by the CASE environment;
- **Technical aspects** *i.e.* what target system representations, in what role (descriptive/prescriptive/instructive), and on what levels of abstraction are supported; and what is the ease and type of use offered by the environment for these descriptions.

A case in point is that the introduction of CASE environments will have a substantial effect in all three aspects, whereas the research in the past has mainly focused on the last one. Examples of effects that may result from introducing a CASE environment are:

- **new organizational forms and strategies** for systems development including building up a new distributed physical environment for system development, enhanced use of physical capabilities such as video-technologies, decision rooms etc (the organizational aspect);
- **new modes and types of interactions** for systems development which include automatic dialogue protocols for review, acceptance and decision-making, enhanced communication capabilities within the development group (semi-structured e-mail), or ease of communicating through a shared description base (the communicational aspect);
- **new target system representations and their manipulation** to enhance creative and skillful systems development. This includes the ease of deriving new versions of target system descriptions, the ease of their verification and validitation, and tutorial support in using various methods (the technical aspect).

Very little is known of these effects and they have been scarcely studied. The most researched area seems currently to be the last one, though most of the results obtained are not outcomes of a substantial and systematic scientific study (*cf.* Chikofsky 1988; Chikofsky & Rubenstein 1988). Therefore more systematic studies are needed before a more solid understanding has been obtained how CASE environments will change systems development in all three areas.

6.2. The Role of CASE shells

CASE environments offer a promise for "programming" the system development process in a truly different way than achieved by proposing a methodology that does not have any computer support. As not all systems development is similar, there is a continual need for adjustment and "reprogramming" the CASE

environment. Therefore, the concept of a CASE shell seems to be very central in adjusting and "reprogramming" the systems development process. The discussion of the different components of a CASE shell provides a starting point to examine in more detail the scope and depth of "reprogramming" offered by various environments. Several observations can be made:

- **an extensive modifiability of description languages** is needed if one wants to affect in a deep sense the technical, organizational and communicational aspects of the CASE environment.
- **the CASE shell must provide means to change the functionality of the CASE environment** *i.e.* it must offer high-level constructs to program the CASE environment.

Thus, the functionality and the linguistic power of CASE shells are essential if any deeper variability in CASE environments is hoped for. We believe also, that this is what we urgently need, as the majority of the methods that are supported by CASE environments have been inherited from the 70's (including structured methods, ERA-modelling etc.). Yet, their usability and support in developing new types of applications (e.g. office information systems) may be inadequate. This necessitates that new methods should be easy to integrate into the existing environments and it should also be easy to build tools to support their use. (See e.g. Kumar & Welke, 1988.) This has several implications for CASE research:

- how to design and configure CASE environments? This will become an important practical issue which needs to be explored in more depth.
- what methods and tools are needed to map conceptual structures and description languages into CASE shells to generate varying CASE environments?
- how interactions between CASE environments and the system development process must be taken into account in the mapping process? In other words, we must explore in more detail how the functions and description languages offered by the environment affect organizational and communicational aspects of the CASE.
- what functions and mappings can be offered at a certain cost *i.e.* what is the value obtained from using the CASE environment (in a specific way) when compared to its costs (such as usage cost, training cost, implementation cost)?
- how various CASE environments can be used, and how they should be used? What aspects and factors affect how they in fact are used? Are there any innovative and new ways to use CASE environments that have not yet been explored?

Several of these research issues are being currently studied and explored. A research project SYTI (an acronym from a Finnish phrase of "system development support environments") (Lyytinen 1988) focuses on the first issue raised. We believe that this is the most critical issue and it offers many untackled problems that must be solved before the other issues can be touched upon (*cf.* Welke 1988; Sorenson et al. 1988). These include: the power of the "metamodelling" techniques needed to model all aspects of the conceptual structures and description languages, as well as their use, the integration of various description languages and their management into a single environment, and the type and functionality of the CASE shells currently offered to support "metamodelling" and environment generation.

Acknowledgements

Thanks to Heinz Klein, Pasi Kuvaja and (especially) John Venable for useful comments on the first version of this paper.

References

. Bubenko, Janis A., *Selecting a strategy for computer-aided software engineering (CASE),* SYSLAB University of Stockholm, Stockholm (June 1988).

. Chikofsky, Elliot J., "Software Technology People," *IEEE Software,* pp. 8-10 (March 1988).

- Chikofsky, Elliot J. and Rubenstein, Burt L., "CASE: Reliability Engineering for Information Systems," *IEEE Software*, pp. 11-16 (March 1988).
- Dart, Susan A., Ellison, Robert J., Feiler, Peter H., and Habermann, A. Nico, "Software development environments," *IEEE Computer*, pp. 18-28 (November 1987).
- Kumar, Kuldeep and Welke, Richard J., "Methodology Engineering: A Proposal for Situation Spesific Methodology Construction," in *Proceedings of CASE Studies 1988*, Meta Systems, Ann Arbor (1988). Meta Ref. #C8811
- Lockemann, Peter C. and Mayr, Heinrich C., "Information System Design: Techniques and Software Support," pp. 617-634 in *Information Processing 86*, ed. H.-J. Kugler,North-Holland, Amsterdam (1986).
- Lyytinen, Kalle, "A Taxonomic Perspective of Information Systems Development: Thoretical Constructs and recommendations," pp. 3-41 in *Critical Issues in Information Systems Research*, ed. R. J. Boland Jr. and R. A. Hirschheim,John Wiley & Sons Ltd. (1987).
- Lyytinen, Kalle, *SYTI-Project: Research Plan*, University of Jyväskylä, Department of Computer Science, Jyväskylä, Finland (Spring 1988).
- Madhavi, Nazim H., Leoutsarakos, Nikos, and Voulioris, Dimitri, "Software Construction Using Typed Fragments," pp. 163-178 in *Formal Methods and Software Development, Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT) Berlin, March 1985*, ed. G. Goos and J. Hartmanis,SPRINGER, Berlin (1985).
- Penedo, Maria H. and Riddle, William E., "Software Engineering Environment Architectures," *IEEE Transactions on Software Engineering* **14**(6) pp. 689-696 (June 1988).
- Sorenson, Paul G., Tremblay, Jean-Paul, and McAllister, Andrew J., "The Metaview System for Many Specification Environments," *IEEE Software*, pp. 30-38 (March 1988).
- Teichroew, Daniel, Macasovic, Petar, Hershey,III, Ernest A., and Yamamoto, Yuzo, "Application of the entity-relationship approach to information processing systems modeling," pp. 15-38 in *Entity-Relationship Approach to Systems Analysis and Design*, ed. P. P. Chen,North-Holland (1980).
- Venable, John R. and Truex,III, Duane P., "An Approach for Tool Integration in a CASE Environment," in *Proceedings of CASE Studies 1988*, Meta Systems, Ann Arbor (1988). Meta Ref. #C8812
- Welke, Richard J., *Metabase: A Platform for the Next Generation of Meta Systems Products*, Meta Systems Ltd., Ann Arbor, Michigan (1988).