SYSLAB

The Systems Development
and
Artificial Intelligence Laboratory

SELECTING A STRATEGY FOR COMPUTER-AIDED SOFTWARE ENGINEERING (CASE)

Janis A. Bubenko Jr

SYSLAB
University of Stockholm

SYSLAB Report No 59

June 1988

SELECTING A STRATEGY FOR COMPUTER-AIDED SOFTWARE ENGINEERING (CASE)

Janis A. Bubenko jr

Swedish Institute for Systems Development (SISU)
Box 1250, S-164 28 KISTA, Sweden
and
SYSLAB
Univ. of Stockholm and the Royal Institute of Technology
S-106 09, Stockholm, Sweden.

SYSLAB Report Nr. 59
June 1988

ABSTRACT

The purpose of this report is to raise the issue of CASE, in particular the question which actions to take now and which strategy to choose for an organization to "enter" the CASE area. The issue is motivated by the large number of CASE-tools commercially available today. Practically all of them support a particular methodology and a software life-cycle paradigm. Buying such a tool implies the necessity for your organization to switch to that particular tool's methodology. On the other hand, there is the option to develop a CASE tool for your own method. This can be done by using a CASE-"shell", a software environment which provides advanced facilities to build your own tools. Such CASE-shells are beginning to appear on the market. Which alternative to choose depends on many factors. In order to obtain a better understanding of the complexity of this issue and in order to obtain a perspective on the problem, the state of the art and research directions in method deveopment as well as in development of CASE environments are first surveyed. The report also presents a set of general, functional features of CASE tools and oulines a general architecture of a CASE environment. Next, a set of possible strategies, ranging from "wait and see" to a hevy engagement in building your own CASE, are outlined. The concluding parts of this report then discusses the selection of a strategy based on a number of situational factors of your organization. The main conclusion is that serious engagement in application of the CASE technology requires very competent staff and a full commitment from the management.

The state of the art and the research sections of this report includes a large number of references. It can, therefore, also be used as an entry point to further reading and/or to looking for research topics in the CASE area.

Introduction

One of the main questions addressed by this report is "shall we let our information system development methodology determine the tool to use or shall we switch to the methodology offered by some of the commercially marketed tools"? The question is motivated by the large number of CASE-tools commercially available today. Practically all of them support a particular method and a software life-cycle paradigm. Buying such a tool implies the necessity for your organization to switch to that particular tool's method. On the other hand, there is the option to develop a CASE tool for your own method. This is then done by using a CASE-"shell". Such CASE-shells are beginning to appear on the market. Which alternative to choose depends on many factors. In order to get a better understanding of the complexity of this question, and to obtain a perspective on the CASE issue, this report first surveys the state of the art of method development, CASE-tool development, and presents an overview of current research related to CASE. Section 4 offers some thoughts on the problem of choosing a strategy for CASE.

1. Methods - state of the art and trends

1.1 State of the art

The software and information systems engineering is still in a crisis with respect to productivity, maintenance costs as well as to the quality of the design products. Industry's currently most popular methods were created in the seventies (e.g. the well-known process oriented methods SASD, SADT, JSD, ISAC , and the data-driven methods ER, NIAM, and others). Many efforts to compare, feature-analyze, and assess methods can be noted. For instance, the IFIP WG8.1 CRIS conference series [Olle82, Olle83, Olle86] has reviewed a large number of "historical" as well as new and research-type methods and has made an attempt to assess their traits and features. The CRIS efforts, and others as well, show that methods are very difficult to compare and to assess due to differences in their conceptual foundations and frameworks. Work has been initiated to establish a conceptual framework within which methods could be more strictly defined, better understood and, perhaps, compared [CRIS-taskgr, FRISCO].

Within the ESPRIT project realm, a few projects have focussed on frameworks and formalisms (languages) to *describe methods* (e.g. the projects ToolUse [Ryan86], and DAIDA [Borgida87,Jarke88]), and on devising a *common semantic model* by which methods could co-operate and exchange design information (the AMADEUS project [Loucop87]).

We can conclude that there currently exists a considerable interest in better understanding and use of "old" methods and in improving them by co-operation with related methods or by integrating them in order to improve their life-cycle coverage and 'power'. An example of this is the integration of process oriented methods with data oriented methods.

In spite of widespread use of popular methods, research experiments show that use of them often present more problems than solutions. For instance, an investigation by Floyd [Floyd86] shows that some popular methods are based on rather 'fuzzy' concepts, and that reasonably precise guidelines for their use are often missing. Other investigations [Brodie87] show that the practical use (in the US) of more strict methods for the conceptual and logical design stages of the development life-cycle is scarce. In spite of the common awareness of the importance to catch design errors as early as possible, it is estimated that, perhaps, no more than 15% of all professionals ever perform a, so called, "conceptual design" (a design stage where high level, strict and conceptual specifications of the application and the requirements are created). Another illustrative estimate by Yourdon is reported in [Chiko88]: 90% of the professionals are familiar with hierarchical, data-flow diagramming techniques, and about 50% have experimentally used the technique. However, only 10% are expected to use the technique actively.

1.2 Research on methods

Research on new and improved methods are manifested by efforts to bring together approaches from the programming language, database, information system, and AI-communities. A series of workshops have been held with representatives of these areas (see, for instance, two books by M. L. Brodie et al. "On Conceptual Modeling .." [Brodie84], and "On Knowledge Based Management Systems" [Brodie86], which contain many articles and references). Some notable trends in methods research and development are

- increased object-orientation (a contribution from the programming language area)
- increased use of formal techniques also in the very early system development stages
- increased use of deductive and rule-based methods
- gradual adoption of 'knowledge-based' techniques to support the system development process as well as the use of the system

Object orientation implies that an application is modelled in terms of communicating, persistent objects (at different generic levels) (see for instance the AVANCE system being developed by SISU [Bjorn88]). The object oriented style improves flexibility, maintainability, component reusability, and supports decentralized architectures.

The importance of capturing user requirements right has stimulated research in languages for capturing and describing *knowledge of the application domain* (its structure and behavior) and of the information requirements in early development stages. Knowledge representation techniques are now being investigated to capture this knowledge. The domain knowledge is then to be used in other design stages for reasoning about the application, for reusing of conceptual specifications, for semantic consistency and quality checking of designs,etc.

A related trend in methods research is towards *deductive & rule based* approaches (e.g. the ESPRIT project RUBRIC [Rubric87]). The basic idea here is to capture and to *explicitly* express 'business rules and constraints' in a *declarative* style rather than to *implicitly* embed them in processing procedure or transaction descriptions. Closely related to the rule-based approach is the *temporal dimension* of information modeling. The basic idea of approaches which use a temporal dimension is the ability to reason about the state of a system at any point in time (see, for instance, the CIAM-method [Gustaf82], the ERAE-model [Dubois86a], and the DADES-method [Olive86]). Of particular interest is the very complex case when not only the contents of a database changes with time but also the *schema*, describing the database contents and constraints, changes, reflecting changes in the application [Abbod87, Martin87].

The use of *knowledge based techniques in systems development* is closely related to increased use of computer-based support tools and environments for this process. Research is under way how to assist a CASE user to use a method in an effective way, how to assist him in the process of validating and in verification of specifications, etc. (see, for instance, proceedings from workshops and conferences such as [RADC87]).

Obviously, there exists a considerable, multi-faceted research activity in this field. However, little of this research is currently being absorbed, or even observed, by professionals in industry and business. The methods used by most practitioners are of the early seventies. It is, however, most probable that the increased use of CASE-tools will accelerate the use of more advanced methods in practice. It has also been observed that the effort to build CASE tools for own "in-house" methodologies and to apply the tools to real projects in the own organization, creates in many cases a deep insight in the potential of such tools, and stimulates research on further extending and developing the methodology as well as the tool. The rest of this report will deal with CASE.

2. General functional features of CASE-tools

In the following, we will denote a CASE-tool a *software environment*, that assists a systems analyst and designer in the process of designing, specifying, analyzing, and maintaining a software product (an information system). We assume the tool supports a particular methodology, covering substantial parts of the systems life cycle. First we will discuss some general, methodology independent, features of such tools. Not all tools on the market today possess all these features to a substantial degree. Next, we outline a crude, general architecture of a CASE tool. The purpose of this section is to present a general framework to use when looking at specific tools.

2.1 General features

The major task of a CASE tool is to accept different kinds of specifications, analyze the specifications, transform specifications, and maintain a large, ever growing set of interrelated specifications. Furthermore, a good tool must also give various kinds of support and guidance to its users. The term *specification* refers generally to *everything a designer is supposed to input to the tool according to the particular methodology and its specification languages*. We partition the functional features of a CASE tool as follows.

User interaction

The most striking feature of most tools today is their user interface, permitting the designer to work with graphical, form-based and textual input/output, often in a multi-window mode. It is difficult to state some general requirements concerning the user interface, except that it should be easy to enter specifications, and that inputting one kind of specification normally requires the concurrent display of (parts of) several other specifications or access to other information (e.g. data term catalogues, concept dictionaries, etc.). It should be possible to enter specifications in an incremental and fragmented fashion, i.e. starting with "skeleton" of a specification structure and then adding details whenever appropriate. The ability to work with menu-driven and windowing techniques seems an important feature, but there is no evidence that this is the only and the best way to interact with a CASE tool.

Concerning output, it should be possible to *selectively* present fragments, and skeletons of (projections of) the specifications (in graphical, tabular and textual form), directed by user queries and projection directives. Browsing, and if appropriate, hierarchical navigation in the specification structure, should be possible. The ability of a CASE tool to *generate documentation*, including graphs, forms, and tables, of selected parts of the specification is an important feature.

Verification support

The purpose of verification is to ensure that a specification is complete and formally correct. *Syntax checking* is an obvious feature of most tools. The next level of verification is *checking of the semantic consistency* of a specification. Here several levels of ambition can be envisaged, such as: checking the specification for violation of static constraints according to the *method's conceptual schema*, checking of the derivability of information, checking for contradictory specifications, and checking of the (intended, correct) behavior of the system according to the behavior expressed by the specification. As, from a theoretical point of view, complete verification is not possible, CASE tools normally adopt a pragmatic approach, and seek practical, partial solutions to the verification problem which are reasonably cost-effective.

Validation support

By *validation* we mean the issue - "does the specification really reflect the user needs and his/hers intended statements about the application?". It can be expected that the most CASE languages and methods, and in particular those with a declarative and deductive flavour, may require a support mechanism which assists the designers in *formulation* of valid specifications as well as in correct *interpretation* of specifications. For instance, the following kinds of support can be envisioned

a) *paraphrasing* of graphical (formal) specifications in natural language (NL)
b) paraphrasing of formal, logic-based rules an constraints in NL
c) *generation of abstractions and abstracts* of (parts of) specifications
d) support for *reasoning about the specification*, e.g. answering "what if X?", "how does X affect Y?", and "why X?" type of queries directed at an existing specification.
e) animation or simulation (or symbolic execution) of (parts of) a specification
f) assistance in *classification and concept formation* (requires domain knowledge), etc.

Design support

In this set of features we include support for transformation of specifications from one 'level' to another. Examples are: *view integration*, *restructuring of a specification*, and transformation of a non-executable requirements specification into an *executable specification for prototyping* (unless the requirements specification language already is already executable). View integration is required to combine the local specification efforts of a number of work teams working in parallel. Restructuring implies the semantics preserving rearangement of a specification in order to improve it according to a set of quality, performance, or other kinds of rules, or according to a designer's restructuring directives.

Development project management

In this category of features we include the following features:

- maintenance of design decisions and design history
- communication and mail facilities
- annotation management
- authorization management

- version and configuration control
- tracking support, change management
- support for co-working in a decentralized design environment

These features are essential for making a CASE-tool work effectively in larger, decentralized projects.

2.2 Outline of a generalized CASE tool architecture

The gross architecture of a CASE tool is shown in the figure below. The heart of the tool, as in all CAD systems, is the "Design Data Base" (DesDB). The DesDB contains several "layers" of data, explained below. A user, i.e. a designer, interacts with the DesDB via an *I/O and presentation* subsystem. Several other subsystems provide support to the user in accordance with features discussed above. The architecture is open-ended, and additional support subsystems should be possible to add (and existing ones improved) as knowledge about them is extended and further developed. The possibility to communicate with other CASE tools must, of course, be provided.

The design data base

The *method's object schema* defines the concepts of the methodology's language(s) in terms of kinds of specification objects, relationships, attributes, and permissible operations on them. It also includes rules and constraints for a complete and consistent specification in accordance with the language and method at hand.

The *method knowledge* part of the DesDB contains information that is used to support a user in using a particular method developing a specification. It contains, for instance, advice to be given in different situations, rules for checking a specification for quality flaws, etc. This part of the DesDB is, for most tools, relatively small, but it is exoected to grow as more experience and method knowledge is acquired.

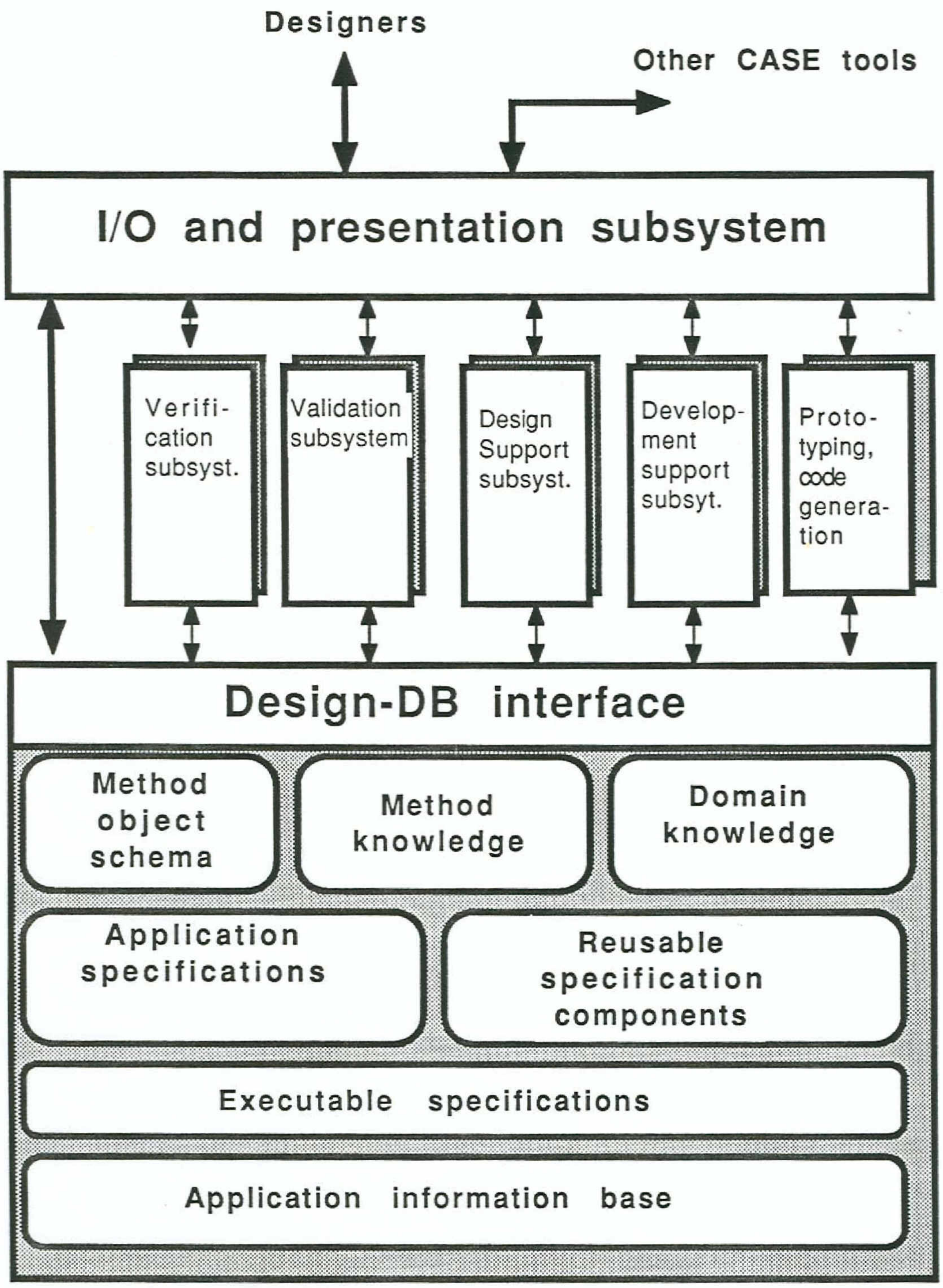
Designers
Other CASE tools
I/O and presentation subsystem
Verifi-cation subsyst.
Validation subsystem
Design Support subsyst.
Develop-ment support subsyt.
Proto-typing, code genera-tion
Design-DB interface
Method object schema
Method knowledge
Domain knowledge
Application specifications
Reusable specification components
Executable specifications
Application information base

Figure: The gross architecture of a CASE tool

The *domain knowledge* part of the DesDB serves to assist the specifier to use appropriate application concepts when developing a specification. This part is also used for more advanced types of semantic and quality checking of the specification. More advanced checking can normally only be done in the context of a particular domain.

The *reusable specifications* part of the DesDB indicates a possible future extension to a CASE Environment. The ideas is to capitalize on components of "older" specifications so that new specifications can be designed by utilising, combining and extending a "libraray" of conceptual components.

The *application specification* is a valid *instance* of the method object schema, developed by designers for a particular application. Clearly, several versions of specifications (or subspecifications) and several application specifications must co-exist and be maintained. An application specification is designed using the various support subsystems of the CASE tool, and using information in the above mentioned parts of the DesDB.

Executable specification denotes specifications generated from the application specifications. They are executable, i.e. the application can actually be run on the CASE tool using the prototyping facility (which interprets the executable specifications). The *information base* denotes a valid instance of the application specification's data base schema.

The support subsystems

The various support subsystems for *verification, validation, design,* and *development* as indicated in the architecture are according to the features discussed in the previous section. As pointed out before, they should form an open-ended system, which can be gradually improved and extended when more experience and knowledge is gained.

3. CASE - state of the art and trends

Work on computer-aided environments, or rather tools, to support some of the life-cycle tasks of systems development methods can be traced back to the fifties. However, not until a few years ago, thanks to greatly improved processing, storage, and graphics interface capabilities of relatively cheap work-stations, the CASE area has experienced an exponential growth in products as well as in research prototypes.

The state of the art in this field can be described in three parts

- development of *method-specific* environments, i.e. tools geared to a *particular* method or chain of methods
- development of *customizable* environments, i.e. environments which can be 'programmed' to support a particular method, or chains of methods

- *research efforts* in this field, mainly addressing some particular analysis or design support problem of a method (or a class of methods)

Common to most of these efforts is the assumption (see previous section) of a *Design (object) Data Base* (sometimes called the 'encyclopedia'). The design data base holds information about the specification objects, their characteristics, and relationships. The *kind* of specification objects stored by the data base is *dependent on the methodology*. **A methodology is thus partly defined by defining a conceptual schema of its design data base.** *Method-specific tools* have the schema predefined for that particular method. *Customizable CASE environments* assume the tool user to define the schema in accordance with his particular method, and to "program" the CASE shell to behave in accordance with requirements of the particular method. Other important components of CASE tools are facilities for performing checking of specifications, facilities for giving 'intelligent' support to the tool user, etc. Research efforts in the CASE area are mainly directed at these latter issues.

3.1 Method-specific CASE tools

Currently there are more than 100 commercial method-specific CASE tools on the international market [Martin88]. Typical products are IEF, IEW, BLUES, DEFT, EXCELERATOR, etc (see [Brodie87, DAISEE'87] and IEEE Software, March, 1988 issue for surveys and tool presentations). The price range of these products is from a few hundred US$ to several tens of thousands of US$. These tools present a great variation of life cycle coverage and tool functionality. Some of them are nothing more than simple diagramming tools (for data modeling, process decomposition, etc), while others support a wide range of design an analysis functions around a comprehensive design data base.

Methods employed by these commercial tools are mainly approaches from the mid seventies. Most tools support some kind of data-flow diagramming approach (e.g. the SASD, SADT, and similar approaches), and some kind of ER-like data modeling method approach. There are great differences in their expressive power, and in their user interface designs and capabilities. Many of the tools have acceptable, or even impressive, user interface architectures, but in general, the *expressive power* of the tools is primitive. Few of them can handle more advanced conceptual modeling concepts such as generalization, constraint definitions, and complex objects. More advanced ('intelligent') verification and validation support facilities are normally lacking.

Experience of use of tools in projects of realistic size is still quite limited. Few tools can report industrial use in LARGE and complex projects. Therefore, neither tool vendors nor tool users are yet sure what functionality they really should expect from a CASE tool.

3.2 Customizable environments - CASE shells

A customizble environment can be denoted a "CASE-shell" (in analogy with an expert system shell). A CASE *shell* includes mechanisms to define a CASE *tool* for an arbitrary method or chain of methods. In order to define a method one must define a design object schema (a conceptual schema of the design objects constituting a particular method, including constraints, derivation rules, operations & preconditions). In addition one must define a set of interrelated and interacting analysis and development work processes by which a *well-formed instance* of the design object schema is gradually developed. Description of the work processes should include guidelines, rules for checking the consistency, completeness, and quality of the designed objects and their relationships. Few commercially available CASE shells exist today, To our knowledge only ECLIPSE by SSL, UK (developed as part of an ALVEY - project), corresponds to our definition of a 'shell' (perhaps also Meta Software's Design OADS, but we lack detailed information at the moment). The CASE-shell area is still mainly a research area. The RAMATIC research effort by SISU, reported in [Dahl85] and in [DAISEE'87], aims at a *generic*, i.e. customizable, graphics oriented modeling support environment.

3.3 CASE research

A very large number of research efforts currently exist in this area. Most of them are directed towards improving the "intelligent support" facilities of CASE-tools for particular methods and approaches by the use of KBS (Knowledge Based Systems) techniques. Typical topics, covered by this research, are

- conceptual specification of methods (e.g. [Reiner84], [Coelho85], [Gustaf86], [Jarke88]). The aim of these efforts is to arrive at strict specifications of methods (their types of objects and work procedures), and, in the long run, of the "knowledge" guiding the work according to a particular method.

- support for capturing of requirements, application knowledge, and assistance in building conceptual specifications. Here we can point to the Knowledge-Based Requirements Assistant (KBRA) effort [Harris88], the Knowledge-based Specifications Assistant (KBSA) effort [Johns88], as well as form driven approaches (e.g. [Manni86]), and natural language driven approaches (e.g. [Kerst86, Cauvet87, Bouze85]).

- conceptual schema diagnosis and restructuring (e.g. [Eick84, Eick85, Cauvet87, Wohed87, Bouze85, Sundin85]. The problems tackled here concern the problem of improving the quality of a conceptual schema specification.

- view integration support (e.g. [Batini86, Johans87]). The problem here is to provide support for integrating local "user views" - local conceptual specifications or schemata - into a global specification.

- assistance in working with methods which employ hierarchical decomposition of processes and data flows in business or real-time, embedded systems. Some researchers (e.g. [Pietri87, Lubars86]) report on prototypes, which can provide support and suggestions to the designer in decomposing a data flow model. Other, ongoing research tries to develop design metrics to diagnose and assess designs for quality, "normality" (i.e. to detect whether a design deviates from what could be considered as "normal"), etc. [Budgen88].

It has been observed in all of the above efforts that "domain knowledge" plays an important role to be able to perform the tasks sufficiently well. Interesting steps towards incorporating domain knowledge have been taken, but much remains to be researched.

A large number of efforts in building "environments" can be noted in the programming language and methodology field. Work here ranges from natural language specification of programs (see for instance [Balz85]), and transformation of specifications to programs, to 'knowledge-based' high-level program editors (e.g. the Programmers Apprentice project at MIT [Waters85]). Some of this work, e.g. the paraphrasing of formal specifications to natural language specifications (for validity checking), may become highly relevant to the CASE approach as well.

Many research contributions and tools exist to provide support in the logical and physical design stages of the systems development life-cycle. Brodie [Brodie87] surveys several tools which provide support for tasks such as transaction analysis, logical/physical database design and code generation.

4. Choosing a strategy for CASE

4.1 The situation

The situation in the methods development area as well as in the CASE area is clearly turbulent and unstable. New directions in methods can be envisioned (section 1.2). It is today difficult to say which direction will dominate, or when we will approach some kind of methodology maturity and consensus, if ever.

The CASE tool market is exploding in terms of number of tools introduced in the market. Many of the commercial tools seem to be reasonable products, but still more seem to be "toys", not suited for use in projects of realistic size and complexity. The growth of the commercial CASE tool market is estimated at about 50 new tools per year. Practically all of them are geared to a specific methodology. Nobody knows how many CASE tools are actually in use today in realistic projects. Probably not too many. The number of research prototypes in the CASE area, being developed at universities and research centres, is larger than the number commercially available tools, and is rapidly increasing.

A particular concern is the difference between the "state of the art" methods employed by today's popular tools and the directions modern software methods research is taking. Will the existing CASE tool vendors change their methods approach and follow the scientific developments? Or will they defend their investment and oppose the introduction of new methodologies? If many more designers start to employ their CASE tools, this will undoubtly further conserve the use of existing methods.

Every organization experiences problems of various kinds (productivity, performance, quality) with their information processing system development and operation activities. It is now only natural these organizations ask the question "can help be obtained from going CASE?", and, having obtained affirmative answers from CASE vendors, further ask the question "which strategy should we choose for introducing CASE in our organization ?".

This complex question depends on many situational factors in your organization. In the following we will examine some possible strategies, and then suggest a number of situational factors which might be considered in selecting a road of action. Finally we will discuss how these factors may influence the actual choice of strategy for CASE.

4.2 Strategies

Below we list a number of possible strategies for dealing with the CASE issue today. Clearly, additional strategies may be formed by selecting a mixture of them.

A **Wait and see.** Observe others while they make all the initial mistakes.

B **Limited experimentation.** Acquire one or two CASE-tools which suit your way of working and which are reasonably compatible with your methodology. Allocate a reasonable budget, sufficient time, and **competent people** to experiment, and to learn more about the CASE area. Make a serious effort to apply the tools to a set of small, but real, projects.

C **Go for a method specific CASE tool.** Make a serious assessment of a number of commercial CASE tools by performing well designed and well planned experiments on small, but "tough" cases. Allocate sufficient resources and **competent people** to these experiments. If the experiments turn out successful, acquire the tool you think is best for you and then make a **seroius commitment** to introduce it in your organization. Otherwise, if the experiments do not turn out successful, you will have learned a lot and you will be able to make better decisions for the future.

D **Build your own CASE tool.** Make an assessment of existing CASE shells and shell vendors. Acquire a CASE shell and develop your own CASE tool geared to your own methodology. Before doing this learn as much as possible about the more powerful, existing method specific tools, related to your methodology.

E **Order your own CASE tool.** Let a shell vendor, using a CASE shell, tailor a CASE tool for your methodology.

F **Integrate several tools.** Try to integrate several CASE tools by making them, if possible, cooperate. For instance, there may be a tool good at conceptual data modeling, and another one good at logical data base design, assuming as input a conceptual data model.

G **Experiment with research prototypes.** Establish cooperation and a joint project with a research organization specializing in CASE and work together with it in experimenting with new methods and tool prototypes.

4.3 Situational factors

In this discussion we will assume a medium size "typical" user organization with its own professional staff for developing and maintaining its computer applications. We, thus, exclude consultants, data processing service centers and similar organizations as their relationship to CASE can vary greatly.

Characteristics of your information system development organization

One important factor is **your organizations policy and attitude** concerning professional development of its staff. Another one is the degree of **realism in its expectations** concerning CASE. The following questions are examples one might ask in order to try to determine your organization's professional maturity with respect to entering the CASE area.

- Is the attitude of your organization's management in favour of making a serious commitment in a CASE experiment without expecting "magic" and immediate results?

- Do you have staff with a good professional and theoretical competence in methods or do you have the potential, knowledge and environment to acquire and train such staff?

- Do you give your staff regular further, advanced training in computer science, methods, and in CASE related topics? Do you encourage your staff to read international, professional periodicals on these topics? (check how many of the references given in this report are familiar to your professionals).

- Do you have a 'central', competent method development and advisory group with sufficient authority and resources?

Characteristics of your method environment and policy

Another factor is the methodological maturity of your organization, i.e. whether you make a serious effort to work according to good 'engineering principles' and standards. Affirmative answers to, for instance, the following questions, might indicate a good methodological maturity.

- Do you have your own *strictly defined and actively used* (manual) system development method that covers substantial parts (stages) of the systems life cycle?

- Do you enforce the use one particular methodology and documentation standards, or is it permitted, within your organization, to work with (possibly fragments of) a multitude of methods?

- Do you strictly enforce project planning, control, cost & performance measurement, and quality assurance procedures in your information system projects?

- Are you actively working to extend your knowledge and your methodology to cover additional sectors/problems of the information systems development area (such as for instance information administration, database administration, security management, systems maintenance, etc)?

Vendor and/or consultant profiles

An important set of factors in selecting a strategy for CASE is obviously the expected relationship between your organization and the CASE vendor, or vendor representative. Consider, therefore, the following.

- Availability of and access to CASE vendors/representatives
- Methodological competence of vendors or vendor representatives
- Vendor potential and plans for future development of tools and methods
- Relationship of your methodology, if any, to the methods offered by tool vendor

4.4 Discussion and recommendations.

We start this discussion with a few dont's. We will then examine the various strategies above and, in particular, comment on the choice between buying a method specific tool and developing your own tool using a CASE shell.

- Do not select strategy A "Wait and see". You will then probably be out of competition and out of qualified staff by 1995.
- Do not believe a CASE tool can compensate the lack of skills and inferior method knowledge of your staff.
- Do not believe that the tools we see today will last - a few of them will be greatly changed and improved and therefore survive. The rest will disappear.
- Do not develop your methodology first and then look around for CASE tools to fit your method. Such tools do not exist. Stop developing "new" methods without considering existing, method specific tools or considering the capabilities of existing CASE shells.

Strategy B, "**Limited experimentation**", seems to be the *minimum strategy* one can choose today. You may choose some of the less expensive (and less comprehensive) tools and experiment with selected method-stages of the systems life cycle. This, of course, requires the support of your management and allocation of sufficient skilled staff, time and resources. If you do not have qualified staff available, you have to consider further training of current staff or recruiting of new staff with the required background. As this is a limited commitment, vendor access and support should not be of utmost importance.

Strategy C is "**Going for a method specific CASE tool**". As this implies a considerable commitment in terms of people, methods, and ways of working, you should have a good methodological maturity and the full backing of your management. Most likely, you must be prepared to switch to another method, or at least considerably change your current way of working. You must be prepared to allocate skilled people to the project and give them the authority and resources needed to enforce the new methods and the new, probably more disciplined, way of working. This strategy requires good access to tool vendors or their representatives. It also requires that the vendors/representatives have the potential to make a serious commitment to your project and serve you with qualified advisors and tutors regarding the use of the method as well as regarding the use of the tool.

Building your own CASE tool, i.e. strategy D, is not recommended if your organization does not satisfy the preconditions for this undertaking. The same conditions concerning management support and backing as for strategy C hold. In addition there is the absolute need for a **strict** and comprehensive methodology, a very qualified methods-staff, and a highly experienced and skilled software staff for "programming" the CASE shell according to your methodology.

Strategy E, "**Order your own CASE tool**", is similar to strategy D, except that you order a qualified vendor or consultant to "taylor" a CASE tool to fit your own methodology. The requirements for management support, resources, and commitment are the same as for D, but in this case you do not need your own software builders to the same extent as for strategy D. On the other hand, the requirement for having highly skilled methods people is just as important. The requirement for a strict methodology is, of course, mandatory. On the other hand, you should not be surprised when working with building a CASE tool for your methodology, the methodology will most likely, more or less, change during the tool building process. The reason for this is that CASE tool building requires the method's constructs to be strictly and formally defined. This implies that during the process of building a tool you will perform a very thorough examination of your method's existing concepts and principles. This examination more often than not leads to changes, improvements, and extensions of various kinds. The same phenomenon, of course, also holds for strategy D.

Integrating several tools (strategy F) may turn out to be a good strategy to follow if you have tools available on the market some of which satisfy your requirements for method support in different stages of the systems development life cycle. This strategy puts very strong requirements on the qualifications of your methods staff as well as on your software people. Research activities in this area are ongoing. However, so far no practical applications of this strategy are, as far as the author knows, reported.

Experiment with research prototypes (G) can be seen as a complementary activity to some of the strategies above. This strategy should be of particular interest to the "more open" strategies D and E, where you are building your own method environment. By studying - and copying - ideas and principles from the research prototypes you may gradually extend and improve your methods as well as your tools. An example of this might be the extension of your tools by incorporating in them "intelligent support" - a theme favoured by many research efforts (see section 3.3).

In conclusion we will examine some general pros and cons of acquiring method specific CASE tools vs using CASE shells to build tools for your own methodology. In a way this gives additional comments on the problem of selecting a B or C strategy vs selecting a D or E strategy. Basically the problem is analogous to developing your own pay-roll system or buying a standard pay-roll package. Below we will suggest some advantages of selecting a method specific CASE tool, and building your own tool using a CASE shell, respectively. The disadvantages of one approach are often the inverses of the advantages of the other.

Method specific CASE tools, assuming you do have your own methodology which is different from the one offered by the tool, should offer the following advantages.

+ **Shorter lead time**. You can install the tool almost immediately and start preparing yourselves to use it and to learn its methodology. Building your own tool will require, depending on its complexity, a lead time of at least 6 months. However, you must also consider the time required to introduce a new method of working in your organization. This may take several months too.

+ **Robustness and performance**. It is probable that a vendor product designed for competition with similar products on the market is well-egineered with respect to robustness and performance. It is also probably more "bug-free" than the tool you build yourselves.

+ **Vendor support**. For marketing reasons a vendor CASE tool must have an adequate organization for providing support to its users in terms of maintenance, new releases, training material, etc. If you develop your own tool, a similar support organization must exist or be established.

Building your own tool using a CASE shell, assuming you have your own methodology, may have the following advantages.

+ **Acceptance**. It is probably more easy to get at tool accepted by your system design staff, if it supports a methodology established in your organization. Less time and resources for re-training of your staff, for development of new working directives, manuals, handbooks, etc., are required.

+ **Changeability and extensibility**. Probably few method specific tools are comprehensive enough to cover other tasks than those typically included in a software life-cycle model. Tasks like information strategy planning, information resource management, data and database administration, quality assurance, security management, and that like, are becoming important activities in organizations, concerned not only of developing specific systems, but also in planning for an integrated and rational use of information technology in a total, organizational perspective. This will require CASE tools to be integrated with, or extended to, tools to support the above activities. Building your own CASE environment offers probably better possibilities to design and implement these extensions according to the need of your organization.

+ **Vendor method independence**. Perhaps the most important factor in choosing between these alternatives is *your judgement* concerning the importance of being independent of future developments, or the possible **lack** of developments, of your "method vendor". We all know of the advantages and disadvantages of being dependent on the hardware and basic software of a selected computer vendor. Buying a vendors method specific CASE tool implies committing yourself to his plans for future method development, if any, and will create extremely strong ties between your organization and the vendor. You have to consider what actions you will take if the vendor switches to a new methodology, and stops further improving the tool you bought, or if the vendor gets out of business.

5. Concluding remarks

Even though many of the available CASE tools of today are "toys", entering the CASE area should be taken seriously. In this article we have tried to illuminate the state of the art of methods as well as of method support tools. We have also briefly surveyed research directions concerning methods as well as concerning tools. Even if we are only at the beginning of the CASE era, knowing little of what requirements to state for good and effective CASE work environments, it is necessary to start learning more about them in order to become qualified "consumers" of the future. We have listed and discussed some possible strategies for dealing with the CASE issue.

Clearly, there are more variants of them, and - clearly - there are more situational factors to consider. However, one of the main messages of this report is that CASE technology will not solve your method and system development problems unless you are prepared to seriously consider the need for further professional development of your staff. According to a survey made by SEI, Pittsburgh [Hump88], the capabilities and the "software process maturity" of the majority of organizations surveyed, are surprisingly low. Organizations must improve their "excellence" in the fundamentals of information and software engineering before making a heavy investment in CASE.

ACKNOWLEDGEMENT

Valuable comments an suggestions to a draft of this report have been given by Alfs Berztiss, Pittsburgh, and Stig Johansson, VOLVO CAR, Göteborg.

REFERENCES

Abbod87	T. Abbod, K. Brown, H. Noble: Providing Time-Related Constraints for Conventional Database Systems, in [VLDB87].
Balz85	R. Balzer: A 15 Year Perspective on Automatic Programming, IEEE Trans. on SE, Vol.11, Nov.,1985.
Batini86	C. Batini, M. Lenzerini, S. B. Navathe: A Comparative Analysis of Methodologies for Database Schema Integration, ACM Computing Surveys, Vol. 18, No. 4, Dec. 1986.
Bjorn88	A.Björnerstedt, S. Britts: AVANCE: An Object Management System, SYSLAB Working Paper Nr 124, Univ. of Stockholm, 1988.
Borgida87	A. Borgida, M. Jarke, J. Mylopoulos, J.W. Schmidt, Y. Vassiliou: The Software Development Environment as a Knowledge Base Management System, MIP-8710, Fakultät fur Mathematik und Informatik Universität Passau, 1987.
Bouze85	M. Bouzeghoub, G. Gardarin, E. Metais: Database Design Tools - an Expert System Approach, Proceedings of VLDB-85, Stockholm, 1985, Morgan Kaufmann Publ. Co.
Brodie84	M.L.Brodie, J. Mylopoulos, J.W. Scmidt (editors): On Conceptual Modeling - Perspectives from Artificial Intelligence, Databases, and Programming Languages, Springer, 1984.
Brodie86	M.L. Brodie, J. Mylopoulos (editors): On Knowledge Base Management Systems - Integrating Artificial Intelligence and Database Technologies, Springer, 1986.
Brodie87	M. L. Brodie: Automating Database Design and Development, Tutorial, Nov. 1987, Authors address: GTE Laboratories Inc., 40 Sylvan Road, Waltham, MA, 02254, USA.
Budgen88	D. Budgen, M. Marashi: KBS Techniques Applied to the Assistant of Software Design - The MDSE Advisor, Univ. of Sterling, UK, 1988. Also in Proceedings of the International Workshop on KBS in Software Engineering, University of Manchester Institute of Science and Technology (UMIST), March 7-9, 1988.
Cauvet87	C. Cauvet, C. Proix, C. Rolland: A Knowledge Based Information Systems Design Tool, Univ. of Paris 1 and 6, France (to appear in Proceedings IFIP TC2/TC8 Working Conference on "The Role of AI in Databases and Information Systems", Canton, China, July, 1988, North Holland).
Chiko88	E. J. Chikofskij: Software Technology People Can Really Use, IEEE Software, March, 1988.
CRIS-taskgr	The CRIS task group (part of the IFIP WG8.1, led by T.W Olle) is about to publish a report, presenting a Reference Framework for Information Systems Methodologies (1988)

Coelho85 H. Coelho, A. Rodrigues, A. Sernadas: Towards Knowledge Based Infolog Specifications - A Case Study of Information Engineering, Decision Support Systems 1, 1985, pp. 143 - 166.

Dahl85 R. Dahl, D. Eriksson, L-A Johansson, U. Sundin, H. Torbjar: RAMATIC - Modeling Support System, SYSLAB Univ. of Stockholm, SYSLAB Report Nr. 34, 1985.

DAISEE87 Data-Stöttet Systemutvikling - DAISEE'87 (Computer-Aided Systems Development), Proceedings from a Workshop arranged by the Department of Computer Science, NTH, Trondheim, 12-13 Nov., Oslo, 1987.

Dubois86a E. Dubois, J. Hagelstein, E. Lahou, F. Ponsaret, A. Rifaut, F. Williams: The ERAE Model: a Case Study, in [Olle86].

Eick84 Ch. F. Eick, Methoden und rechnergestützte Werkzeuge für den logischen Datenbankenentwurf, PhD Thesis, Univ. of Karlsruhe, Fakultät für Informatik, Karlsruhe, FRG, July, 1984.

Eick85 Ch. F. Eick, P.C. Lockemann, Acquisition of Terminological Knowledge Using Database Design Techniques, Proc. ACM SIGMOD 1985, pp. 84-94, 1985.

Floyd86 C. Floyd: A Comparative Evaluation of System Development Methods, in [Olle86].

FRISCO87 IFIP WG 8.1 has in the fall of 1987 initiated a task group FRISCO (FRamework of Information System COncepts) with the objective to improve the conceptual foundations of information systems.

Gustaf86 M.R. Gustafsson, B. Wangler: A Conceptual Model for Computer Supported Information Systems Development, SYSLAB Workning Paper 113, 1986.

Gustaf82 M. R. Gustafsson, T. Karlsson, J. A. Bubenko Jr: A Declarative Approach to Conceptual Information Modeling, in [Olle82].

Harris88 D. R. Harris: An Overview of the Knowledge-Based Requirements Assistant, in [RADC88].

Hump88 W. S. Humphrey: Characterizing the Software Process: A Maturity Framework, IEEE Software, March, 1988.

Jarke88 M. Jarke, M. Jeusfeld, T. Rose: Modelling Software Processes in a Knowledge Base: the CASE of informations Systems, Report, Univ. of Passau, P.O.Box 2540, D-8390 Passau, F.R. Germany, 1988. Also in Proceedings of the International Workshop on KBS in Software Engineering, University of Manchester Institute of Science and Technology (UMIST), March 7-9, 1988.

Johans87 B-M Johansson, C. Sundblad: View Integration - a Knowledge Problem, SYSLAB Working Paper 115, 1987.

Johns88 W. L. Johnson: Overview of the Knowledge-Based Specification Assistant, in [RADC88].

Kerst86 M.L. Kersten, H. Weigand, F. Dignum, J. Bloom: A Conceptual Modelling Expert

System, 5th Internat. Conf. on the ER-approach (ed. S. Spaccapietra), pp. 275-288, Dijon, France, 1986.

Kung83 C. H. Kung: An Analysis of three Conceptual Models with Time Perspective: in T. W. Olle et al (editors): Information Systems Design Methodologies - a Feature Analysis, North Holland, 1983.

Loucop87 P. Loucopoulos, W. J. Black, A.G. Sutcliffe, P.J. Layzell: A Unified View of Model Representations in System Development Methods, Department of Computation, UMIST, P.O. Box 88, Manchester M601QD, U.K.

Lubars86 M. D. Lubars, M.T. Harandi: Intelligent Support for Software Specification and Design, IEEE Expert, Winter, 1986

Manni86 M.V. Mannino, J. Choobineh, J.J. Hwang: Acquisition and Use of Contextual Knowledge in a Form-Driven Database Design Methodology, 5th Internat. Conf. on the ER-approach (ed. S. Spaccapietra), pp. 141-157, Dijon, France, 1986.

Martin87 N. Martin, S. Navathe, R. Ahmed: Dealing with Temporal Schema Anomalies in History Databases, in [VLDB87].

Martin88 C. F. Martin: Second Generation CASE Tools: a Challenge to Vendors, IEEE Software, March, 1988.

Olive86 Antoni Olive: A Comparison of the Operational and Deductive Approaches to Information Systems Modeling, IFIP Congress 1986, North Holland, 1986.

Olle82 T.W. Olle, H.G. Sol, A.A. Verrijn-Stuart (Editors): Information System Design Methodologies: a Comparative Review, North Holland, Amsterdam, 1982.

Olle83 T.W. Olle, H.G. Sol, C.J. Tully (Editors): Information System Design Methodologies: a Feature Analysis, North Holland, Amsterdam, 1983.

Olle86 T. W. Olle, H. G. Sol, A.A. Verrijn-Stuart (Editors): Information System Design Methodologies: Improving The Practice, Elsevier Science Publishers B.V. (North Holland), Amsterdam, 1986.

Pietri87 F. Pietri, P.P. Puncello, P. Torrigiani, G. Casale, M. Degli Innocenti, G. Ferrari, G. Pacini, F. Turini: ASPIS: A Knowledge-Based Environment for Software Development, ESPRIT Project 401, Proceedings ESPRIT Technical Week, 1987 (see also an article about ASPIS in IEEE Software, March, 1988).

Raupp84 T. Raupp: Qualitätsverbessernde Transformationen Konceptueller Schemata, Diploma Thesis, Univ of Karlsruhe, Fakultät für Informatik, FRG, 1984.

Reiner84 D. Reiner, M. Brodie, G. Brown, M. Friedell, D. Kramlich, J. Lehman, A. Rosentahl: The Database Design and Evaluation Workbench (DDEW), IEEE Database Engineering, Vol. 7, No. 4, Dec. 1984.

Rubric87 ESPRIT project 928: RUBRIC - A Rule Based Representation of Information Systems Constructs.

Sundin85 U. Sundin: A Plan Generating System For Conceptual Schema Restructuring:

SYSLAB Report Nr 33, Chalmers Univ. of Technology, Göteborg, Sweden, 1985 (to appear in H. Kangassalo (Ed) "Information Modelling and Database Management", Springer-Verlag)

RADC88 Rome Air Development Center: Second Annual Knowledge-Based Software Assistant Conference, Proceedings, Rome Air Development Center, Air Force Systems Command, Griffiss Air Force Base, NY 14331-5700.

Ryan86 K. Ryan (Editor): An Experimental Basis for ToolUse, Technical Report CSC-86-3, Dept. of Computer Science, Trinity College, Dublin 2, Ireland, 1986.

Waters85 R.C. Waters: The Programmers Apprentice: a Session with KBEmacs, IEEE Trans. on SE, Vol. 11, Nov., 1985

VLDB87 P. M. Stocker, W. Kent (editors): Proceedings of the Thirteenth International Conference on Very Large Data Bases, Brighton, UK, Morgan Kaufmann Publ. Inc. 1987.

Wohed87 Rolf Wohed: Diagnosis of Conceptual Schemas, SYSLAB, Univ. of Stockholm, SYSLAB Report Nr. 56, 1987, (also to apperar in Proceedings IFIP TC2/TC8 Working Conference on "The Role of AI in Databases and Information Systems", Canton, China, July, 1988, North Holland).

THE SYSTEMS DEVELOPMENT AND ARTIFICIAL INTELLIGENCE LABORATORY

The Systems Development and Artificial Intelligence Laboratory (SYSLAB) is a joint venture between the Department of Computer Sciences at the Chalmers University of Technology and the University of Göteborg (SYSLAB-G), and the Department of Information Processing and Computer Science, at the Royal Institute of Technology, Stockholm and the University of Stockholm (SYSLAB-S). The laboratory is chiefly sponsored by STU (The National Swedish Board for Technical Development).

SYSLAB research covers the following research areas.

1. Conceptual modeling of applications including modeling languages, knowledge representation, modeling environment and database design.
2. Artificial intelligence and its application to development and use of information systems.
3. Interactive and distributed systems including office information systems and distributed databases.
4. Graphics based environments for general model management and design.

For further information:

SYSLAB-S
Dept. of Information Processing & Computer Science
University of Stockholm
S-106 91 STOCKHOLM, Sweden
Phone: 46-8-162000

ISSN 0281-6164