INSTITUT FÜR INFORMATIK

UNIVERSITÄT
STUTTGART

Institut für Informatik, Azenbergstr. 12, 7000 Stuttgart 1

Azenbergstr. 12
Herdweg 51
7000 Stuttgart 1
Telefon (0711) 2078-
Telex TX 07-21703

Gerhard Fischer and Heinz-Dieter Boecker:

THE NATURE OF DESIGN PROCESSES AND HOW COMPUTER SYSTEMS CAN SUPPORT THEM

appeared in: Proceedings of the European Conference on Integrated, Interactive Computer Systems (ECICS), Stresa, Italy, 1982; to be published by North Holland, Amsterdam

THE NATURE OF DESIGN PROCESSES AND HOW COMPUTER SYSTEMS CAN SUPPORT THEM

Gerhard Fischer and Heinz-Dieter Boecker
Research Group on Man-Machine Communication
Computer Science Department, University of Stuttgart

Abstract

One of the major obstacles to develop powerful computer systems that support design processes is an insufficient understanding of how humans solve design problems. We describe an epistemology of design based on theoretical and empirical work in different domains (software engineering, creative writing, technical construction systems, learning environments). In all these domains design is best understood as an incremental activity that makes use of existing prototypical solutions to gain a deeper understanding of a problem.

1. Introduction

Our long range research effort is to improve man-machine communication through the creation of computer systems in which the human and the computer cooperate to solve problems and achieve tasks more quickly and more efficiently than either of them could do working alone. We are in the process of designing and implementing a personal, integrated, knowledge-based Information Manipulation System (**INFORM**) (this work is part of a research project supported by the German Ministery of Research and Technology; see BOECKER, FISCHER and GUNZENHAEUSER 1980). In **INFORM** we construct prototypical computer systems which support design processes in different domains (software engineering and office automation). The goals of the research described in this paper are threefold:

1) to characterize and understand the fundamental principles of design processes
2) to derive requirements for design support systems
3) to describe systems which meet some of these requirements.

Many design problems have demonstrated to us that there are inherent limitations to the complexity that the unaided designer can control in any situation.

2. Epistemology of Design

2.1 Definition

Design is concerned with how things ("artifacts") **ought to be,** in order to attain goals and to function whereas the natural sciences are primarily concerned with how things **are.** Designers try to shape the components of new structures. In most cases the goals are only specified as predicates to be fullfilled and not as detailed plans to be simply executed (see ALEXANDER 1964 and SIMON 1981).

SIMON (1981, p109) uses the work of an architect to characterize the process of design: "Architecture can almost be taken as a prototype for the process of design in a semantically rich domain. The emerging design is itself incorporated in a set of external memory structures: sketches, floorplans, drawings of utility systems, and so on. At each stage in the design process, the partial design reflected in these documents serves as a major stimulus for suggesting to the designer what he should do next. This direction to new subgoals permits, in turn, new information to be extracted from memory and reference sources, and another step taken toward the development of the design".

Determining the boundaries between domain specific and domain independent design knowledge, understanding what a designer does and what kind of difficulties he encounters are necessary preconditions to develop computer systems which give better support to the designer than pencil and paper.

2.2 Concepts of Design

Computer science has (re-)invented a set of well known (eg top-down, bottom-up, stepwise refinement) and less known (eg use of stepping stones, progressive constraints, structured growth, metadescription, context-sensitive descriptions, filters) concepts of design. Most of them are relevant for design tasks in many domains and should be taught explicitly.

To validate these concepts we applied this terminology to design tasks in a totally different domain: **design with technical construction systems.** At the same time we tried to extend the epistemology of design with new insights gained from working in a different domain.

2.3 Creating (Synthesis) and Understanding (Analysis) a Design

Crucial processes in creating a design are (see quotation from Simon in 2.1):

1) to deal with sets of possible worlds (ie we want to consider design alternatives)
2) to incorporate the emerging design in a set of external memory structures (ie we need a description system that will support the information gathering process)
3) to record the important parts of the design path (and not only the design product)
4) to create low-cost modifiable models which help us to replace **anticipation** (of the consequences of our assumptions) by **analysis** and **inspection** and allow us to experiment with a prototype; in principle, theories could be used to create and understand our artifacts, but exposing a product (whether it is a program, a scientific paper or a model built with FT) to experiments generally augments our understanding of it (cf. NEWELL and SIMON 1976).

Understanding (analysis) of existing designs is crucial for several reasons:

1) it is a prerequisite for modification (which often becomes necessary because it is impossible for the designer as well as for the potential user to foresee all future uses of a complex system)
2) the insufficiencies of existing prototype models have to be uncovered; the surface manifestations of problems have to be traced back to their causes.

Regarding design as an incremental activity whose phases are interlaced implies that there is no strong seperation between synthesis and analysis. The synthesis process is repeatedly interrupted to analyze what has been done so far and to find the best way to proceed.

2.4 Design Conflicts

The major problem in many design tasks is to cope with conflicting goals or to be precise: to cope with conflicting methods to reach different goals. Among others we have explored the following conflicts in detail:

1) cognitive efficiency versus machine efficiency in programming
2) simplicity of a design or a design tool (so that it can be easily handled by a large group of people) versus power (so that we can construct a large variety of different and complex systems)
3) the necessity to remain compatible with existing systems (eg timesharing computers) versus exploiting the possibilities of new media (eg personal machine environments)
4) ease for first time users versus convenience for fluent users: mnemonic names in an editor help the first time users but they quickly get into the way of an experienced user; the conflict can be resolved by constructing adaptive computer systems
5) tight integration between different subsystems (eg like in Smalltalk or Interlisp) versus flexible, general, reconfigurable modules or tool fragments (like in UNIX)

In many situations we are not even aware of the real design issues. For a long time designers of computer systems have considered the amount of information which can be generated as a relevant design criteria whereas empirical evidence has shown that the scarce factor in dealing with complex systems is **human attention.** This insight leads to quite different design criteria like intelligent summarizing, efficient chunking methods and creating with the help of filters views of reduced complexity into complex information structures.

2.5 Satisficing versus Optimal/Correct Solutions

To label a solution as optimal or correct makes sense in simple tasks that are precisely specified (eg of Factorial or Euclid's algorithm); we can determine an optimal and correct solution by comparing the implementation to the specifications. But for most design tasks we do not have complete and detailed specifications and thus cannot search for the best alternative (which we are unable to recognize anyway unless we have generated all alternatives). Usually, it is more realistic to define criteria for an acceptable or **satisficing** solution, which is "based on decision methods that look for good or satisfactory solutions instead of optimal ones" (SIMON 1981, p 139).

3. Design in Different Domains

To gain a better understanding of the processes which we want to support we have studied design processes in several domains:

- software engineering (BOECKER, FISCHER and GUNZENHAEUSER 1980)
- creative writing (FISCHER 1980)
- technical construction systems (FISCHER and BOECKER 1981)
- learning environments (FISCHER 1981)

We have verified and extended the basic set of thought processes underlying design which serve as guidelines for the implementation of design support systems.

3.1 Software Engineering

Our understanding of software engineering as a design activity is different from formal specification methods. The later require a level of understanding which we do not have for most systems. For many design problems there is no other way than to plunge into activities which are not fully understood. If our understanding is limited or if we are

lacking detailed specifications of the final goals several alternatives have to be developed and compared with each other.

Specification through rapid prototyping therefore is the most promising methodology for this class of problems. It allows us to produce a large set of potential designs that are cheap enough to be thrown away and detailed enough to be tested and to be argued about. Rapid prototyping is especially crucial in information technology, because it has close to zero fabrication costs (ie the costs to produce an arbitrary number of copies of a finished software product can be almost neglected).

There are many ways for computer systems to support rapid prototyping: more automation (to free the human programmer from the clerical tasks), more assistance (to exploit the strong parts of the human information processing system, eg visualization and intelligent summarizing), more self-knowledge (to elucidate the conceptual structure) and large system capacities for the development process (application systems can be produced through down-loading).

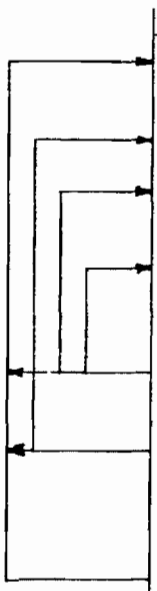
In our work we are not so much concerned with the construction of individual algorithms but with computer systems which support the development of complex systems. It is typical for many design problems that a system consists of small building blocks whose fundamental properties and laws of behaviour are well known. The main difficulty of the design problem, however, resides in predicting how an assemblage of a large number of such components will behave. A differentation between the two domains is given in Fig. 1.

Figure 1: "Algorithms" versus "Systems"

	Algorithms	Systems
examples	Factorial, Fibonacci numbers	text processing systems, expert systems (MACSYMA, Dendral)
specifi-cations	precise, complete; do not change over time	not precise; incomplete; change over time; evolve over time
use	small building block; in general only useful within larger systems	real world systems; to fit an environment of needs
local transparency	intricate logic; highly optimized, require proof	locally transparent; complexity is based on a large number of interactions between modules
classificat. in PS terms	well-structured problem, abstract problem	ill-structured problem, problem in semantically rich domain
design criteria	efficiency, correctness	modifiability, transparency uniformity, reliability
understanding	can be comprehended as a single "gestalt"	comprehension without adequate tools is almost impossible
semantics	intrinsic	extrinsic

Our system should support all steps in the design process and not only the execution of algorithms and the coding activity (see Fig. 2).

Figure 2: Steps in Design Processes



Steps	Associated tasks
problem formulation	recoding of private experience; "understanding" of the problem
specification	formalizing the problem
planning	defining the subproblems; finding the right representations
implementation	description in a programming language
execution	investigation of the dynamic behavior
verification, testing	comparison between specification and implementation
documentation	recode the design path, the teleological semantics, retains the past
maintenance	adapt programs to changing environments
modification	adapt the program to new purposes, predict the non-local effects of changes

It should reduce the time which is needed to develop a first draft of a design into a useable system. It is more important and more difficult to find the "right" design (ie the problem specification should match the problem itself) than implementing the design and showing the correctness between specification and implementation. The specification process is more complex and evolutionary than previously believed (see quotation from Simon in section 2.1) and this raises the question whether the predominant view of a specification as a fixed contract between a client and a designer stands up against reality. Specifications (as the already fixed) and implementations (as the yet to be done portions of a multi-step development) should not be totally separated, because in realistic situations they are intertwined (SWARTOUT and BALZER, 1982).

3.2 Creative Writing

It is perfectly acceptable in creative writing (GREGG and STEINBERG 1980; FISCHER 1980) to have a first draft which is considered not to be the final product but serves as a stepping stone for a better version. For an essay as well as for a scientific paper it is evident that criteria for satisficing solutions and for understandability are more appropriate than the notion of a "correct" solution (as opposed to what conventional programming methodologies propose).

Empirical evidence has demonstrated that good text processing systems which make the planning (which is not restricted to a linear medium any more) and editing process a much less painful and time-consuming task can cause a qualitative improvement of the documents produced. The possibility to produce many more drafts, which can be

objected to criticism make rapid prototyping a realistic methodology.

3.3 Technical Construction Systems

Besides computer related design processes we have studied theoretically and empirically design processes with technical construction systems. Our goals were:

1) to show the similarity of design processes in different domains
2) to verify and extend the epistemology of design
3) to show the usefulness of the computational metaphor in a totally different domain.

We have chosen FISCHERTECHNIK (FT) as the specific technical construction system to work with. FT can be enjoyed by "children of all ages". It allows the designer to construct - starting with a universal building block - very simple models (eg a car, a crane etc) as well as completely functional models (eg a technically exact model of an assembly line for car manufacturers). FT is not only an excellent toy (it gained the "Oscar de Jouet" in 1972 for being elected the best toy of the year) but a realistic tool which allows building designs of complexity comparable to big software systems. FT is a good example of a design tool that has **"no threshold, no ceiling"**: easy things can be easily done and difficult things are possible to be done.

3.4 Design of Learning Environments

The designer of a learning environment has to pay attention to many different factors (eg the skill to be learned, the environment, in which the skill is executed and the equipment used). The crucial problem is that the starting state and the goal state are too far apart. Our design task is to find adequate microworlds (through appropriate simplifications) which allow us to guide the learner from the initial to the final form of the skill (for details see FISCHER 1981).

4. Extended Epistemology of Design

4.1 Empirical Findings

Our empirical research based on working in different domains verified some of our intuitions, namely

1) some design processes (see section 2.3) are basically the same in whatever domain they exhibit themselves; making the tacit knowledge of designers more explicit will improve our ability to describe, explain and teach the process of design as well as support it with more adequate tools;

2) design is an incremental activity; a partially completed product can be used to gain a deeper understanding of the problem (with respect to constraints and possible solutions);

3) most artifacts (especially software systems) must be continually modified to meet changing requirements; no amount of initial design work is a complete substitute for actual use; therefore the construction of convivial systems (see 4.3) is no luxury but an absolute necessity;

4) the major difficulty in design tasks is not to understand the intrinsic semantics of individual building blocks but to predict how an assemblage of such components will behave;

5) it is important for the understanding of an artifact (which is a prerequisiste for modification) that we have access not only to the final product but to the key parts of the evolutionary path which led to it;

6) the language developed in Computer Science and specifically in AI is adequate for the description of design processes in different domains.

4.2 The Impact of Different Design Media

Limitations in human information processing capabilities (eg the limited short term memory, our inabilities to predict the consequences of our assumptions or to follow long chains of reasoning steps) have led to the development of support systems that augment human intelligence (eg pencil and paper, wind tunnel, technical construction systems but also formalisms ("technology for thinking") like the arabic numerals, musical notation and programming languages).

The impact of technology on design can be characterized as follow:

1) technology speeds up the analysis-synthesis cyles to produce new models approximating the solution;

2) enlargement of strategies (eg a text processing system frees us from producing a document in a linear fashion);

3) resolving some of the mentioned design conflicts (see 2.4);

4) providing efficient chunking methods (eg representing information on a computer screen in a way that we can make use of our visual system);

Fig. 3 describes different design tools and environments according to the advantages and disadvantages which they offer. We are convinced that the computer can be (the statements in Fig. 3 indicate the desired and not the currently achieved properties) an ideal tool for design -- if we develop the right systems for it (see section 5). Currently most computer systems are worse than pencil and paper: they cannot be used as a two dimensional medium, they enforce a teletype-oriented style of communication and they are not easily available yet.

4.3 Convivial Tools

Illich (1973) has introduced the notion of "convivial tools" which we regard as important for design support systems. He defines them as follows: "Tools are intrinsic to social relationships. An individual relates himself in action to his society through the use of tools which he actively masters, or by which he is passively acted upon. To the degree that he masters his tools, he can invest the world with his meaning; to the degree that he is mastered by his tools, the shape of the tool determines his own self-image. Convivial tools are those which give each person who uses them the greatest opportunity to enrich the environment with the fruits of his or her vision.
Tools foster conviviality to the extent to which they can be easily used, by anybody, as often or as seldom as desired, for the accomplishment of a purpose chosen by the user."

Illich tries to point out alternatives for future technology-based developments and their integration into society. Applying his ideas to information processing technologies and systems shows that conviviality is a dimension which sets computers apart from other communication technologies. All other communication and information technologies (eg television, videodiscs, interactive videotex, video games) are passive, ie the user has little or no possibilities to shape them to his own taste and tasks. He has some selective power but their is no way that he can extend system capabilities in ways which the designer of those systems did not foresee.

Figure 3: Advantages (+) and disadvantages (-) of different design media

Real world:
+ no abstraction processes are required
+ tests are reliable (they are carried out with the real thing)
- tests are difficult to carry out (controlled experimental situations are hard to generate)
- only one level of description; simplification or abstraction is very difficult
- costs may be very high (especially if existing structures have to be modified)
- phenomena have to be explained or predicted with all their particularities

Technical construction systems (Fischertechnik):
+ many natural laws are valid (eg an artifact constructed the wrong way will fall over); it is possible to explore the world by providing a means to act on it concretely
+ the dynamics of models can be studied
+ costs are relatively low compared to the construction of prototypes in the real world
+ undoing things is easy
- in general there is only one additional level of description (compiling is restricted)
- resource limitations are crucial (number of available parts; diversity of parts)
- models can not be parameterized

Paper and pencil:
+ arbitrary many levels of descriptions
+ no resource limitations with respect to material; but we hit the boundaries of human rationality and imagination very quickly
+ cheap to modify; easily available
+ diagrams help to visualize structures and experiments
- hard to modify with respect to time (see advantage of text processing system)
- static, no dynamic behavior; we cannot dynamically recode the appearance of a structure
- the laws of nature are not valid (eg things do not fall over; experiments with nature are impossible)

Computer:
+ ease of modification
+ compiling allows us to change a glass-box into a black-box (ie we can generate several levels of description)
+ the dynamics of a model can be simulated
+ almost no limitations which systems can (at least potentially) be modelled
+ UNDO can be provided for ease of experimentation
+ models can easily be parameterized
+ irrelevant details can be hidden
+ the natural laws can be modelled; we can even model worlds which do not obey the laws of physiscs
- the natural laws are not valid
- the modelling requires an additional level of abstraction
- non-trivial use requires (at least today) special knowledge

We do not claim that current existing computer systems are convivial. Most systems belong to one of the following two classes which constitute extreme cases relative to the ease of use and the amount of control they offer to the user:

- turn-key systems: they are easy to use, no special training is required but they can not be modified by the user

- general purpose programming languages: they are hard to learn, they are often too far away from the conceptual structure of the problem to be solved and it takes too long to get a task done or a problem solved.

There are promising ways to bring these extreme cases together and to make systems more convivial. Good turn-key systems contain features which make them modifiable by the user without a necessity to change the internal structures. There are text processing systems which allow the user to define his own keys, abbreviations and macros and there are display systems which allow the user to create and manipulate windows at an abstract and easy to learn level (BAUER, BOECKER and FISCHER, 1981). Another idea related to the concept of convivial systems is a **"toolkit"** which provides a set of components and set of tools (by means of which these components can be viewed and manipulated) that can be used to create different but related things. This approach has been successfully exploited by technical construction systems. Examples in the world of information processing systems are developments like SMALLTALK and UNIX which allow the user to create complex systems not by writing large programs from scratch, but by programming through specialization or by interconnecting relatively small predefined components. LISP is also a good example for a convivial tool, because -- based on the equivalence of data and programs -- it is easy to write programs which manipulate other programs.

With the advent of convivial tools old distinction will break down: there will be no sharp border line any more between programming and using programs -- a distinction which has been a major obstacle for the usefulness of computers. Convivial tools will take away the impossible task from the "meta-designer" of design-tools that he has to anticipate all possible uses and all people's needs.

5. INFORM — a System to Support Design Processes

The goal of the project **INFORM** is to build an information manipulation system (IMS) which supports its user in his information processing needs. With respect to the methods and techniques used it is at the boundary of AI and software engineering and focusses on the central aspects of man-machine-communication. Our goal is to increasingly integrate the components of **INFORM** until finally there will be only small conceptual and technical differences between writing a program, composing an essay, constructing a database or any other design task which the user wants to carry out with the help of a computer. Important design criteria for this integration are two dimensions of uniformity:

- linguistic uniformity: all tools (eg the programming system, superimposed modules, more specific creations of the user) are made from the same material. This has the sociological benefit that the system's implementor and users share the same conceptual world which contributes to the overall conviviality of the system. Each module in the system is a **"glass-box"** that the user can modify and inspect to its edges.

- uniformity of interaction: The crucial aspect of the interface that provides a uniform structure for finding, viewing and invoking the different components of the

system is the use of a display screen, that allows the real-time, direct manipulation of iconic information structures which are displayed on the screen. Each change is instantly reflected in the document's image to reduce the cognitive burden for the user. The screen is regarded as an extension of the limited human short term memory (ie it provides a similar support like pencil and paper for adding two ten digit numbers).

INFORM incorporates the idea of a **symbiotic environment of man and machine** (in the sense of McCRACKEN, 1979) which combines the advantages of both to cooperatively solve design tasks which neither of them could solve as quickly and as easily alone. The human provides goals, integrates different knowledge sources, uses common sense knowledge and solves problems by analogy. The machine serves as an external memory aid, keeps an agenda of things to do, propagates changes, dissects complex information structures into different perspectives and hides irrelevant details. There has been a continous effort in computer science and especially in Artificial Intelligence to delegate more and more tasks to the machine, so the human can say **what** he wants to be done and the machine cares **how** to do it.

We currently concentrate our efforts to support the human in three different areas: general components to improve human-computer communication, software production systems and end user systems (eg for applications in office automation).

5.1 Design Support through Better User Interfaces

Since knowledge based systems are of little use if the information cannot be delivered to the user in a way which takes the cognitive limitations of the human into account research on the bottleneck of man-machine communication, the user interface, is an important issue within the project **INFORM.**

One of the greatest steps forward in man-machine communication was the possibility to use the display as a truly two-dimensional medium. But new inventions introduce new problems. We are now confronted with a new set of **display management problems** that did not arise with the teletype terminal: what information should be diplayed? How should it be displayed? How can we direct the users attention so he will notice an important message? Which new techniques are possible?

It is quite obvious from the knowledge structures in **Fig. 6** that we do not want to just dump out the information structures collected. Instead, we want to make use of a system which supports multiple windows (BAUER, BOECKER and FISCHER 1981) so we can see selected views of the complex structure (see **Fig. 4**). To fullfill the demands of conviviality the **user** should have control to define filters so he can determine the relevancy of the information.

Generating context-sensitive, multiple perspectives each of reduced complexity is a common technique among designers used to dissect otherwise intractable entities (compare the use of maps that picture economic, political and hydrological perspectives of a country). The use of windows supports this approach and allows us to regard a program as a complex information structure, of which only parts are of interest to the programmer at a certain time.

- information about the structure of the program (eg index programs, symbol tables)
- documentation of the tradeoffs of certain design decisions
- overview about the space of possible design decisions
- information related to the dynamic behaviour (eg time and space requirements)
- notes related to the history of a piece of software
- descriptions in natural language.

The knowledge base uses a specialized version of OBJTALK (an object-oriented knowledge representation language (FISCHER and LAUBSCH 1979; LAUBSCH 1982) similar to SMALLTALK, KRL and FRL). The knowledge representation machinery build into OBJTALK provides a good base for the implementation of knowledge bases. The basic frame-like knowledge units are organized around the concept of the function (since we work on LISP and LOGO programs); we plan to extend this to cover information about data and class structures as well as packages of functions. **Fig. 6** shows an example of the knowledge base compiled for a function which is used to compact and cleanup a database in a simple database system.

Most of this context structured knowledge base is simultaneously compiled while the user is working interactively on his problem. It further includes information supplied by the user, eg a purpose slot and a natural language description of the algorithm that may be selectively read and inspected; these slots are not used by the interpreter but they serve an important role in providing **memory support** for the human programmer.

After a function is defined it is immediately translated into the internal representation like the one displayed in **Fig. 6.** Since all further analyses make use of this internal representation it has to contain all information. From this example it should be quite obvious that in a knowledge-based system a program is **more** than its listing.

The knowledge compiled in these knowledge structures can be used to guide the programmer in editing sessions or to give him the possibility to selectively read the code written by somebody else from special point of views.

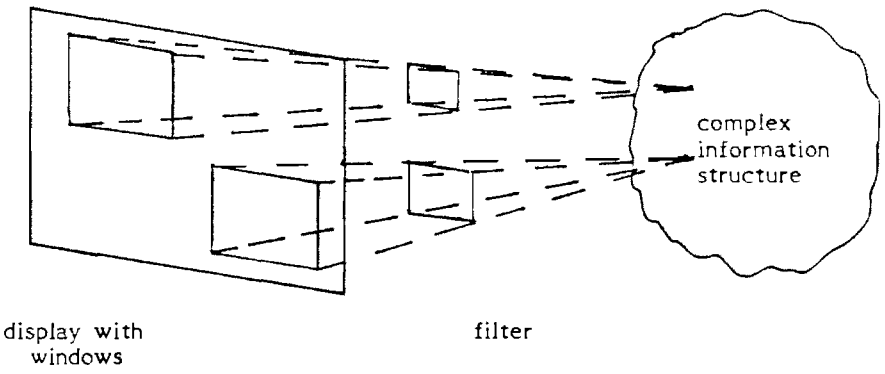
5.3 Design Support in End-User Systems

One of the end user systems which we are developing in **INFORM** is a planning program PLANER which assists students of our institute to plan their graduate studies. With this example we want to study the use of a domain dependent knowledge base to construct a helpful and cooperative tool. The system helps the user to construct a time table for one semester by providing suggestions according to certain specifications (eg no class before 10:00am, at most eight hours a week etc) given by the user. Conflicts that arise are either resolved by the system using a knowledge base of preferences supplied by the user or the user is asked to make a decision. The system infers the implications of decision made by the student in his current time-table for later semester (eg one class may be required as a prerequisite for another class, some courses may be taught only once every two years).

6. Conclusions

SIMON (1981) claims that "the proper study of mankind is the science of design, not only as the professional component of a technical education but as the core discipline for every liberally educated man". Illich points out, that politics in a postindustrial society must be mainly concerned with the development of design criteria for tools rather than with the choice of production goods as it is now.

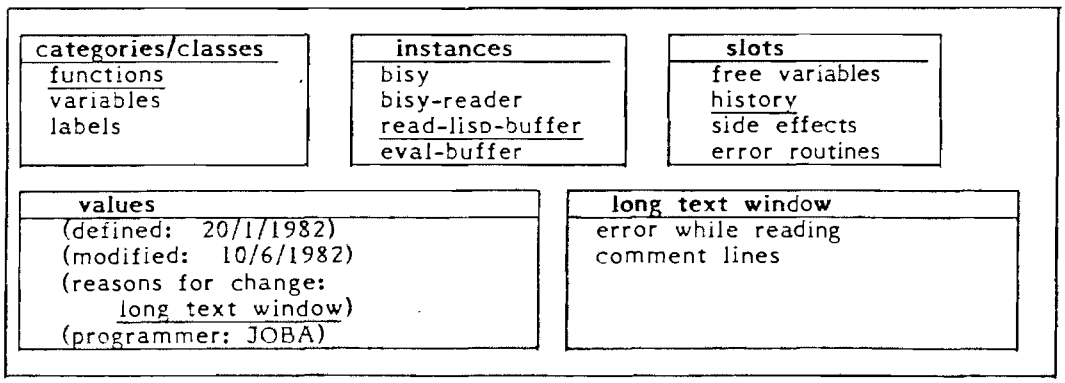
Figure 4: Multiple Perspectives Using Filters and Multiple Windows



A window system is also a prerequsite to implement a good browser (GOLDSTEIN and BOBROW, 1981). A **browser** is a display-based interface which allows a user to examine a complex environment without prior knowledge of its exact structure. It allows the user to navigate in a complex space and to focus on his area of interest. Browsing capabilities can be used to filter information and summarize it into semantic units.

Fig. 5 shows one of our browsers (it displays selectively parts of the information structure which is given in figure 6). With the help of this browser we can traverse a hierarchical network by sequentially selecting items starting in the leftmost top window (selection is shown by underlining). Based on our selection the contents of the next window changes accordingly.

Figure 5: An Example of one of our Browswers



In order to really exploit the human visual system we have further developed visualization support systems that display LISP data structures in graphical forms and provide movie like traces of program execution.

5.2 Software Production Systems

The theory of design as put forth in section 3 serves as a framework for our work on software production systems. We see a program as a complex information structure that includes:

- notes about the teleology of a program

Figure 6: Example of a Knowledge Structure in INFORM

Conventions:
1) **Bold:** slot names of our frame-like structure
2) underlined: user-provided information and commentaries
3) normal font: data supplied by the system
4) CAPITALS: "guesses" of the system, ie derived information which is incomplete and uncertain
5) **Bold** and underlined: inherited information

```
(Function: read-lisp-buffer
 (comment: "BISY means: bildschirm- and syntaxoriented editor;
            eobp means: end-of-buffer-predicate")
 (purpose: "makes a list of LISP-lists and atoms out of the BISY-buffer and
            returns it as a value")
 (algorithm: "conses all characters of the buffer together, applies a
              readlist on them and lets this list begin with a progn)
 (Status: defined)
 (Code:
   (def read-lisp-buffer
        (lambda nil
             (prog2 (progn (push-position) (set-position 1 1))
                    (do ((source nil)
                         (ch (get-char-att) (get-char-att)))
                        ((eobp)
                         (car
                          (errset
                           (readlist
                            (cons "(" (nreverse (cons ")" source)))))))
                        (or ch (setq ch 10))
                        (setq source (cons (char ch) source))
                        (rightc))
                    (pop-position)))))
 (Package: BISY-Userfunction)
 (iscalled by: (bisy bisy-reader eval-buffer)
 (calls:  (as command: push-position set-position rightc)
          (as function: get-char-att char pop-position)
          (as predicate: eobp))
 (type: function)
 (parameters: 0)
 (local variables:      (ch      (type: fixnum)
                                 (used as: (+ ASCII-Code (* 256 Attribut)))
                                 (possible values: from-0-to-65536))
                        (source (type: list-of-fixnums)
                                 (used as: input-stream)
                                 (possible values: all-list-of-fixnums)))
 (free variables: ())
 (history:  (defined:  20/1/1982)
            (modified: 10/6/1982)
            (programmer: JOBA)
            (reasons for change: "error while reading LISP comment lines"))
 (side effects: ())
 (Error routines: errset))
```

We believe that design concepts are the right vocabulary which should be used for software engineering, research in problem solving and expert systems and that they are much more adequate than programming language constructs. Our goals for doing this research were: to gain a better understanding of design (a cognitive science task) and to use this understanding to build systems which support the design process (a cognitive engineering task).

References

Alexander, C. (1964): "The synthesis of form", Harvard University Press

Bauer, J., H.-D. Boecker and G.Fischer (1981): "Entwurf und Implementation eines Systems multipler Fenster", in Notizen zum Interaktiven Programmieren, Heft 4, Oldenburg

Boecker, H.-D., G. Fischer and R. Gunzenhaeuser (1980): "Die Funktion von integrierten Informationsmanipulationssystemen in der Mensch-Maschine Kommunikation", Project Proposal, MMK Memo, Institut fuer Informatik, Universitaet Stuttgart

Fischer, G. (1980): "Cognitive Dimensions of Information Manipulation Systems", in R. Wossidlo (ed): "Textverarbeitung und Informatik", Springer Verlag, pp 17-31

Fischer, G. (1981): "Computational Models of Skill Acquisition Processes", Proceedings of the 3rd World Conference on Computers in Education, Lausanne, North-Holland, Amsterdam.

Fischer, G. and J. Laubsch (1979): "Object-oriented Programming", in Notizen zum interaktiven Programmieren, Heft 2, Stuttgart, pp 121-140

Fischer, G. and H.-D. Boecker (1981): "Understanding Design", Proceedings of the 3rd Annual Conference of the Cognitive Science Society, UC Berkeley, California.

Gregg, L.W. and E.R. Steinberg (eds. 1980): "Cognitive Processes in Writing", Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Goldstein, I. and D. G. Bobrow (1981): "Browsing in a Programming Environment", Proceedings of the 14th Hawaii Conference on System Science.

Illich, I. (1973): "Tools for Conviviality", Perennial Library, Harper and Row, New York

Laubsch, J.: "ObjTalk - eine LISP-Erweiterung zum objektorientierten Programmieren", MMK Memo Nr. 22, Institut fuer Informatik, Universitaet Stuttgart, 1982

McCracken, D. (1979): "Man + Computer: A new Symbiosis", CACM, Vol 22, No 11, pp 587-588

Newell, A. and H. Simon (1976): "Computer Science as Empirical Inquiry: Symbols and Search", CACM, Vol 19, No. 3, pp 113-126

Simon, H. (1981): "The sciences of the artificial", 2nd edition, MIT Press, Cambridge, MA

Swartout, W. and R. Balzer (1982): "On the inevitable Intertwining of Specifications and Implementations", CACM, July 1982, Vol 25, No 7, pp 438-440