Readings on Cognitive Ergonomics –

Mind and Computers

Proceedings of the 2nd European Conference
Gmunden, Austria, September 1984

Edited by
G.C. van der Veer, M.J. Tauber, T.R.G. Green and P. Gorny

INTRODUCTION

Gerrit van der Veer, Michael Tauber, Thomas Green, Peter Gorny

In August 1982 the "First European Conference on Cognitive Engineering" was held in Amsterdam. Scientists from the multidisciplinary domain comprising informatics, engineering and cognitive psychology met, to discuss the theories, methods, models and experiences of their disciplines. Fruitful contacts were made, some of them quite stable. In the next two years this resulted in a better awareness of what was going on in the different centres in Europe. Experimentalists became aware of new theoretical concepts and models, theoreticians collected examples to illustrate their models, scientists involved in design methodology became aware of human factors and developed design criteria to take care of these.

This volume is a tentative step towards integration of these different streams. It is in no way meant to mark final positions. At the "Second European Conference on Cognitive Ergonomics" in Gmunden, Austria in September 1984 the discussion will be continued, as will be done at various other meetings in the second half of 1984. The current contributions should be conceived as working papers, outlining both what has been effected thus far and what tasks are set for the near future.

Theoretical and methodological issues will be dealt with first, followed by contributions from the field of cognitive ergonomics and software design. Applications in the domain of learning, aspects of graphic presentation modes and industrial applications will conclude the overview.

1. MODELS AND METHODOLOGIES

In the art of systems design the notion of model is of growing importance. Models depicting users feature in new design methods, guiding the designer about what behaviour to expect from the future end-user. Meta models may be devised to incorporate both user and system, creating a conceptual framework for communication about design problems. Users will need models of the system in order to predict the outcome of their actions, and in fact always will develop models, whether the designer did implement them or not. The first part of this volume opens with a critical review of the relevant literature concerning modelling, human-computer interfaces and related concepts, from the viewpoint of cognitive psychology and computer science. In this analysis Rohr and Tauber build on Moran's command language grammar and Oberquelle's notions of human-computer communication. They elaborate upon a representational framework as well as present a methodology for building interfaces and teaching people how to use them.

Modelling tools and procedures help the user reduce the complexity of the real system and to influence it. In the view of Oberquelle the most appropriate concept from the user's point of view is the model as a communicable abstract description of the relevant aspects of the system, a "homomorphic image". The learning of a model is facilitated if the intentions of the builder and the user are not too different. Oberquelle develops a meta-model of human computer communication, a dialog between abstract partners as a process, worked out in a three part model incorporating separate roles for the designer, the dialog system and the user. Each of these partners has its own model of the whole communication process. The cooperation

between designer and user is important for the improvement of models of human computer communication. A procedure is proposed in which users are trained in modelling tools (descriptions for external representation, e.g. nets, both graphical and formal), common models of the work situations intended should be developed, and it is suggested that the final implementation should be preceded by prototyping and testing.

Design for information systems no longer being an artistic profession, there is a growing need for the development of methods. Traunmüller categorises the different approaches in devising methods in:
- a procedure oriented approach: analysis and description of hierarchically structured functions;
- a database oriented approach, with static properties;
- a behaviour oriented approach, with dynamic properties like transitions and conditions;
- prototyping as an experimental method;
- logical modelling at the conceptual level ("mental prototyping").

To illustrate the relation between design methods and cognitive ergonomics examples are cited and reviewed on their communicative qualities concerning different target groups: builders, manager, end users. Emphasis is laid on design as a process, not an end product, and on participation of the end user in the early stages of this process.

Van der Veer and Van Muylwijk present methodological notions, illustrated with an example about field studies of the development of knowledge of a computer systems during the introductory phase of learning. The method starts with an analysis of the content of the course, concentrating on the models and metaphors the teacher intends to transfer.
Tests for entering behaviour and the concepts present in the novice have to be developed, their outcome should be compared to measurements of the final models. Relevant cognitive styles may be measured and their effect upon the development of the models in individual users may be established. Descriptive observations and correlational analysis may result in hypotheses about causal relations, to be verified empirically by observing comparable learning situations that vary along well defined dimensions.

2. COGNITIVE ASPECTS

The quality of human computer interaction is to a large extend determined by the ergonomic features of the system. When knowledge about cognitive psychological notions is adequately assimilated in the design the human partner may interpreted the reactions of the systems as "logical" behaviour. On the other hand the user may be allowed to make use of semantic constructions that are consistent with his "natural" knowledge structure.

Communication between human beings makes use, besides natural language, of other aspects like gestures. According to Schneider, Lind, Allard and Sandblad human computer communication can not easily be constructed in this way. Recent models of human cognition feature a distinction between a high capacity subconscious parallel processing system and a sequential conscious limited capacity processor.
Time space patterns are represented in the first system. In constructing human computer interfaces it seems useful to represent as much as possible of the program and its functions as time space patterns. As far as computer functions are designed analogous to those environmental aspects that our sensory, motor and intellectual abilities have been shaped to handle, the system is easy to use, even for beginners.

The interaction with computers consists partly of handling complex descriptions. Potts suggests a canonical formulation for the expression of different specifications for different categories of users and different viewpoints.

PREFACE

The chapters in this book are papers that will be read on the Second European Conference on Cognitive Ergonomics - Mind and Computers, Gmunden, Austria in September 1984, sponsored by IFIP WG. 6.3 (Human-Computer communication), and organised under the auspices of the Östereichische Computer Gesellschaft OCG), the Gesellschaft für Bildungstechnologie (GBT) and the Gesellschaft für Informatik (GI).

Cognitive Ergonomics is a new interdisciplinary research field between computer science and psychology. The investigation of design-criteria for human computer interfaces under cognitive aspects, modelling of systems, users and interfaces and empirical work characterise the research in cognitive ergonomics (sometimes also called cognitive engineering). The aim of this book and of the Gmunden conference is to bring together contributions from European scientists of Cognitive Ergonomics.

In preparing this volume, the editors owe a great deal to Francis Brazier, who carefully read, interpreted and if necessary reworded the contributions of most of those authors whose first language is not English. Generously investing far too much of her time, she succeeded in preserving the original meaning. Elly Lammers was responsible for the organisation involved in this publication, for the lay out and for typing the greater part of the papers. She spent her days telephoning through Europe and her nights communicating with the UNIX system. Without Francis and Elly this book would have lost much in readability.

July, 1984

Gerrit van der Veer
Michael Tauber
Thomas Green
Peter Gorny

CONTENTS

NOVICES AND LEARNING

INTERFACES IN THE FIELD

ORGANISATIONS AND SYSTEMS

The "external interface" comprises the user interface, appropriate to some category of users, between external description and conceptual description. The "conceptual interface" comprises the translation between the task and the internal description according to some viewpoint. Steps in the transformation of descriptions are reformulation, filtering, generalisation and integration. This process requires for a domain model and a user model. PROLOG may be used as a canonical formalisation for specifications. Rapid prototyping is a means of validating specifications by direct observation.

Hoc presents an experimental evaluation of the effect of representation of the data structure. If the program is written in such a way that the commands refer to a semantic representation of the data, incorporating information about the next operation conditional on the effect of the foregoing, the response latencies and error rates reflect preparation of the subjects for the next action. In the case of an abstract representation however, incompatible with the control structure implied in the data, this preparation disappears and the subject is forced to explicitly identify every condition.

The difficulties in learning and using text editors are analysed by Green, covering a design of notation, the choice between many "weak" or few "powerful" commands, the possibilities offered by speech recognition and the origins of users' mistakes. A recurrent theme in this paper is the need to create command language structures that are appropriate to the task and visible to the user.

3. SOFTWARE ENVIRONMENTS

Two papers about meta-communication (communication about the human-machine intercourse, often about problems in interaction) are presented in this section.

Fisher, Lemke and Schwab deal with user support systems incorporating a knowledge base about the problem domain, communication processes, the communication partner and common problems of the users. This communication may take place along implicit or explicit (graphic or menu based) channels. User support functions may be divided into tutorial, explanatory, documentation and help systems. The help facilities are further analysed in passive and active help systems. Requirements for help systems incorporate a view of human information processing, the individual user's expertise and knowledge structure and knowledge about the task domain. An example of a prototype system in given, implemented for the screen oriented editor BISY, both in passive and active modes.

As the traditional error-messages are of little use for novice programmers to debug their first attempts, Du Boulay and Matthew present an analysis of the desired functions of error systems. A survey is given of the existing attempts to built "intelligent" compilers. The authors present a prototype error checker, written in PROLOG, working on a small subset of PASCAL. There are five subsystems, concerned with lexical, syntactic, semantic and logical analysis and a trace mechanism. These subsystems look first of all for those errors that are characteristic of novices' behaviour, and react in a language adjusted to the level of knowledge of a naïve user.

4. NOVICES AND LEARNING

When a person is first brought into contact with a computer system some kind of learning may be expected, and indeed the situation is often deliberately structured in a way that is intended to optimise learning. Optimisation in the contributions in this section is based on different theoretical analyses, in some cases validated by different amounts of empirical evidence.

Analysis of problem solving theories in a paper by Waern points to 5 factors determining speed of learning in interaction with computer systems. Theoretically 4 different types of learning result could be expected: goal-condition-method rules, higher order rules, problem schemata and causal relations. These different concepts can all be incorporated in the idea of a model of the system.
Observations with different tasks, varying along the mentioned 5 factors that should determine learning speed, confirmed the existence of results of the types of goal-condition-method rules and problem schemata. Examples of the other types of learning results could not be detected in these observations. Users rely strongly upon prior knowledge when approaching a new system. Transfer could be negative as was observed in the case of an experienced programmer who spent much time discarding well established schemata.

For the acquisition of textual information in computer assisted teaching Kommers advocates a combination of learner control and "web-teaching" (based on presenting the structure of the learning material in a network in which the configuration and centrality of the concepts are derived from expert knowledge). Adaptation of the presentation sequence to prior knowledge of the student may be achieved by different conversational strategies of which examples are given.

An overview is presented by Van der Veer and Beishuizen, of a long term research project investigating prospects for the application of computers in education, both from a cognitive psychological point of view (evaluated by experimental methods) and from the point of view of actual use and user acceptance (long term observations in primary schools). Combination of both kinds of result leads to suggestions for fruitful applications depending upon both the changeability of cognitive functions and the goals of the learning process.

5. INTERFACES IN THE FIELD

This section contains examples of prototype interfaces for practical situations (Haring et al. and Peruch et al.), followed by two studies in which feature the cognitive ergonomic aspects of visual displays. Falzon comments also on interactive language systems. All of the studies result in suggestions for future applications.

The paper by Haring and Krasser describes a case study of the design of a human computer interface for novice users (staff of a company in mechanical engineering industry) for an information storage and retrieval system. The design process started with an analysis of the goals into primary design goals and human factor design goals. The interface provided several dialogue styles (menu selection, form filling and the selection of function keys) each of which has advantages and disadvantages. The design phases parallel the software life cycle (planning, definition, design, implementation, installation, maintenance). Important in the case described is the integration in the development procedure of a model of the user, to which the output of each step is related. Besides this, rapid prototyping offers another opportunity for validation.

Peruch et al., report a study of ergonomic aspects of dynamic graphical support systems for control tasks (ship manoeuvring), based on a model of the cognitive processes, developed in a multi disciplinary approach. The method used is a simulation of ship displacement due to environmental conditions and actions of the pilot (non professional or professional). Two modes of presentation are compared: cartographic vs. perspective pictorial information, both complemented with alphanumeric data about speed and steering parameters. The results show the perspective mode to be the most efficient, being isomorphic to "natural" processing.

Referring to psychological theories about the input side of human information processing, short term memory capacity and attention switching between different input streams, research is reported by d'Ydewalle on the perception of visual display units, especially the effect of reading subtitles below a dynamic visual information stream on the screen. Eye movement registration revealed characteristic fixation patterns: Subjects commonly look at the moving image first and thereafter spend a short time reading one or two keywords from the subtitle. They jump directly to these words. This suggests a preparation via parallel processing of the verbal input channel. To process the subtitles totally, subjects might need a presentation time that is longer than normally provided in subtitled films.

Falzon presents an overview of studies in human computer interaction at INRIA. The problem of the design of visual representation of information is analysed, presenting alternatives for the same situation that differ in the way relations are represented. This representation should be compatible to the mental representation of the user. Research on interactive language systems for novices focussed on command languages using icons for objects and words for actions. Redundancy in command language turned out to be helpful for experienced users, but not for novices. Again compatibility is important, between the label used in the command language and the concepts evoked in the user by these labels. The same holds for the structure of the software and the semantic notions of the user.

6. ORGANISATIONS AND SYSTEMS.

Some aspects of the impact of the introduction of computer systems in society are analysed in the last division of this volume. The effect on the organisation structure of employees in industry and in a hospital ward are represented by case studies. Some experimental results are reported by Pinsky and Pavard and a case of a national development and validation program for job selection procedures is described.

Two different fictional cases of the effect of computerisation in a manufacturing environment are sketched by Clegg et al. In the first company the end users / operators share the responsibility for tool setting, planning and programming with the official programmers. They enjoy their jobs, earn high wages but on the other hand the management has problems in retrieving information from the shop floor. In the other company, in all respects comparable to the first, management is completely in control. Machine operators have only a monitoring function, tool setting and programming is done by different specialists. The operators show a low level of satisfaction and motivation. Comparison of these extreme opposites shows that the social structure is certainly not determined by technology alone. Strategic choices by the management result in different structures of operational control, accompanied by different patterns of benefits and costs.

Mathiassen and Andersen present a case study of the organisation of tasks between nurses in a hospital ward, showing the changes caused by the introduction of computers in the work situation. It is worth noting that the nurses were actively involved in the introduction of the system and in the definition of the formal language used, the semiotic system. An analysis is made of the effects of the pattern and content of the inter personal communication, on the semantic structure, the expression level

and the content structure of the language used in communication, and on the roles in the organisation.

The contribution by Pinsky and Pavard deals with ergonomic experiments on structure of activity and cognitive processes. A first example illustrates an on line data coding task. The method of prototyping is combined with the analysis of verbal protocols of the users, in order to discover inadequacies of the system. The second example deals with text composition tasks at a word processing system. The method of analysis in this case is the unobtrusive recording of action sequencies in an experimental task comparing different traditional and computerised editing systems.

Pitariu describes a psychological analysis of the prerequisites for predicting success in man machine jobs, and reports validation studies for the selection of test batteries for various kinds of programmer jobs, key punch operators and computer operators. This is part of a national program for the professional training in informatics.

MODELS AND METHODOLOGY

REPRESENTATIONAL FRAMEWORKS AND MODELS FOR HUMAN-COMPUTER INTERFACES

Gabriele Rohr, Michael J. Tauber

IBM Germany Heidelberg Scientific Center
Federal Republic of Germany

This paper is mainly based on discussions between a computer scientist and a psychologist concerning models of human-computer interaction. It tries to specify what a model is, which purpose it serves, and which components of human-computer interaction have to be modelled. Furthermore, these specifications are compared with already existing models.

Models are discussed recently in connection with building up an adequate user interface architecture. Design criteria are needed to construct interfaces which take into consideration human information processing abilities as well as task structures represented by the human. Hereby, "Architecture means the complete and formal description of the surface of a system seen from a well-defined interface. Therefore, architecture is more than the usual specification. Architecture also contains a model of the user and a model of the communication between a user and a system ... Architecture does not refer to the product only. With the same weight, architecture refers to the production process and its documentation" (Zemanek 1982; translated by the authors).

To meet these requirements, models of human behavior in interaction with computer systems are needed. Several models of human-computer interaction have been worked out in the past. They differ however very much in the aspects they describe. A classification of these models has not been done yet.

In this paper an attempt is made to clarify the knowledge and methods that are required to build up an adequate model of human-computer interaction and would help to formulate an abstract architecture. Central roles in the discussion play the terms model and representational framework. It must be pointed out that the first step in building up an architecture is the complete formal specification of the virtual system from an intended user's point of view.

1. MODELS

1.1 Definition of the model concept

A model is a representation of an object where relevant properties of the object are mapped onto a specific substrate different of that of the object (symbolic signs, electrical circuits, etc...). The selection of "relevant" properties presupposes a subject which decides the relevance and a function for which the representation is a purpose. The object can consist of one or more partial objects and their relation to one another, but it must at least contain two clearly definable properties which can be related to each other.

Generally, a model is regarded as embedded in a ternary relationship R (M,S,O), where

- M stands for the model
- S for the subject and his intended model function
- O for the object to be modelled.

Building up a model means to specifiy M, S, and O, and to define the mapping relations between O and M (see Klaus and Liebscher, 1979).

To specify M means to determine the substrate of which the model is to be made, i.e., if it should be material (electrical circuits etc.) or non-material (a special sign notation like mathematics, graphical sign structures, or programming languages, etc.). The selection depends on the purpose of the model.

To specify S means to determine the purpose the model should serve (i.e. to gain knowledge about a structure, function, or behavior of a system (e.g. stability) or to gain design criteria for a machine, etc.).

Finally, to specify O means to determine the components (properties) of the object to be modelled and their assumed relation structure of which parameters have to be identified.

An important point in the use of models is their evaluation concerning their quality. Quality means how well a model can either predict or describe the behavior and the performance, or to describe the structure of the modelled object.

1.2 General Purpose of man-machine models

Man-machine models are generally used to predict overall man-machine systems' performance. Since performance measures do not necessarily include measures of the individual behavioral steps in time history of a human interacting with the machine, only little can be said about the causes of high or low performance (Rouse 1980). Hence, models of human and machine behavior are required. Behavioral models are stronger than performance models allowing statements about causes of performance. Only through such models, detailed criteria for improving performance in man-machine interaction can be developed. This would mean either improving machine structure and/or dynamics on the one side, or improving of human skills on the other side.

In any case, independently of the chosen strategy, a set of parameters relevant for the behavioral control (on the machine and on the human side) has to be defined. The main purpose of man-machine models is to identify these key parameters and their values. After their identification, an explicit model results allowing predictions about man-machine systems's behavior and its performance limits under different structures and dynamics. From these models, design criteria for machines or teaching strategies for improving skills can be derived.

The definition and identification of key parameters can become very difficult, especially if the system to be modelled is very complex. For this reason, only simple man-machine systems have been explicitly modelled as yet with numerical valued parameters, e.g., models of machine control. Manual control systems are mainly defined on the machine side by the amplitude and the time delay of response in the frequency domain, and on the human side, perceptual motor time delays, frequency limits and predictive abilities as key parameters (see Figure 1).

By means of these values, transfer functions can be computed, and overall performance, and stability limits if disturbances are added can be predicted (see Johannsen et al.,

1977). Once analyzed, the system can be improved by training the human controller, or improving the machine characteristics.

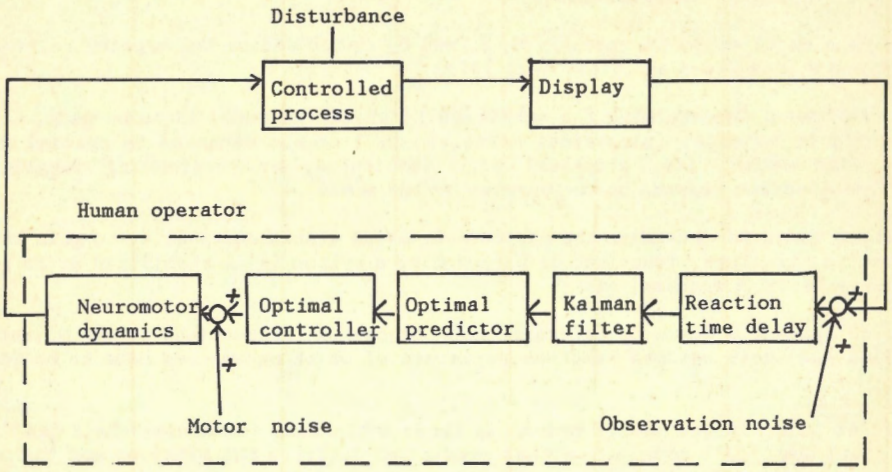


Figure 1: Example of a model of a manual control system

A good example is the system of human ship control. The critical parameter value is the great latency responce of the machine. Together with external disturbances it leads to great instabilities of the system if there is no high predictive ability of the human. Only few people have this high predictive ability for such tasks and training methods are not very efficient. These findings led to the construction of predictive displays (Widdel and Krais, 1982).

Human-computer interaction is not as easy to model as the system described above. First of all, the parameters of the abilities required are not well defined. Secondly, the system is very complex and structured by a symbolic representation. The kind of structuring varies very much.

1.3 Modelling Human-Computer Interaction (HCI)

Modelling the human-computer interaction means clearly to define a set of parameters on the machine side and another set of the human user's side which could be related to one another. It must be noticed, however, that in this case the parameters on the machine side can only be defined with respect to the symbolic representation of the machine, i.e. the interface, and not to the machine structure itself. Because there is a large amount of possibilities of varying interface structures, whose parameters must be regarded in close connection to the parameters of human abilities.

Unfortunately, designers often forget that the user is a system component within the system of human-computer interaction, or they overestimate the learning capacity of the human user.

What is the main concern of a user interacting with a software system (symbolic representation of the computing machine: interface)? The user must build up a mental rep-

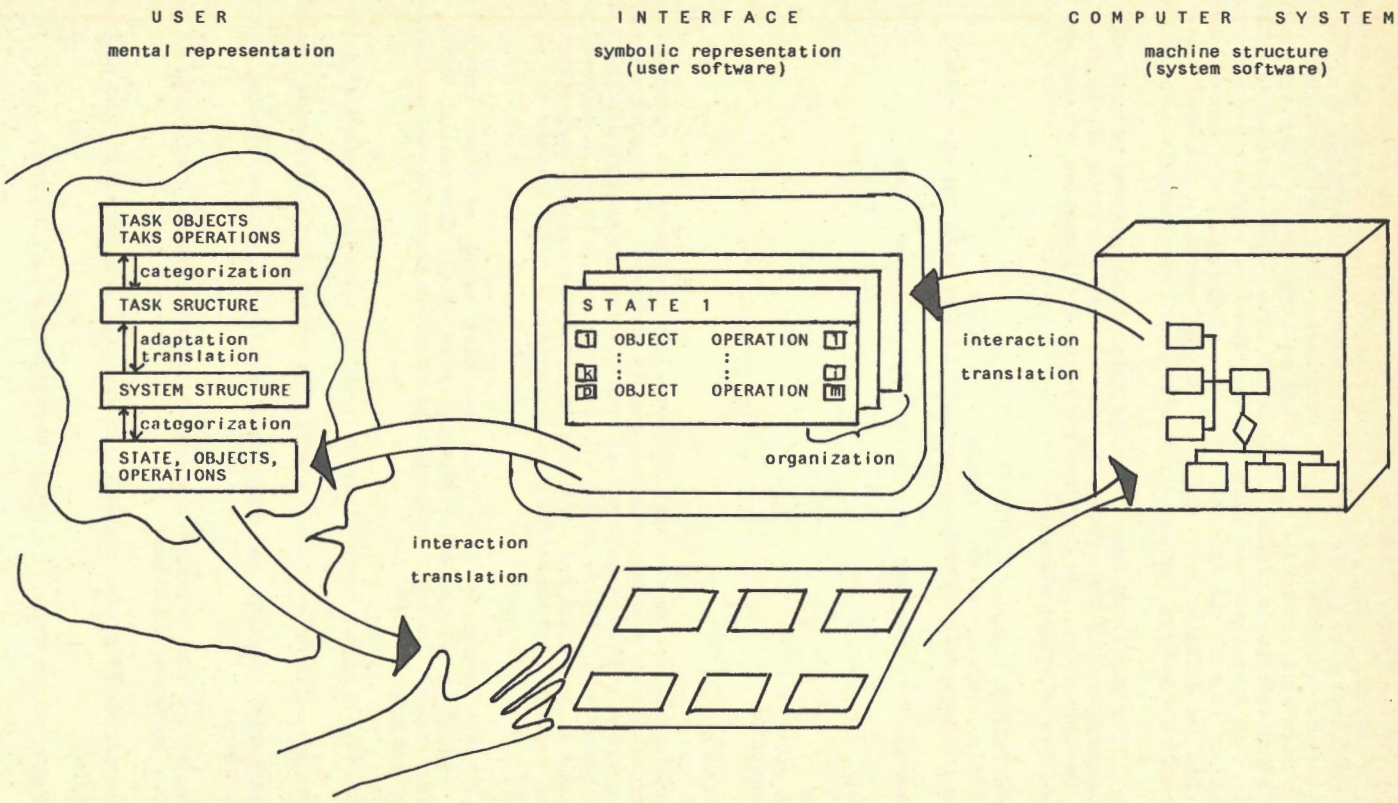


Figure 2: A model of system representations in human computer interaction: Components, connections between and transformations on them

resentation of the system's structure and gain knowledge about the functions of this system with respect to a set of tasks. Furthermore, he must learn the language, i.e., a set of symbols, their syntax, and operations connected to them, to evoke interaction sequences related to task and subtask functions. So, the user's representations of the system structure are models of a virtual machine, whereby "virtual machine" is defined as a representation of the functionality of a system (functional units and their behavior). The most important point for the user is the relation between task and machine, and not so much the internal structure of the machine's system. Consequently, the task for the designer is to model a suitable interface as a representation of the virtual machine which can serve as a possible mental representation for the user.

Summarizing, for modelling human-computer interaction systems we must regard the symbolic representation of the system, the mental representation of the user, and their relation to each other. Hereby, the symbolic representation of the machine system consists of the following elements:

- Objects (things to operate on)
- Operations (symbols and their syntax)
- States (where special operations are possible: structuring operations)

and the mental representation can be structured in representing:

- Objects
- Operations
- States
- System structure
- task structure

(see Figure 2)

Mental workload affecting the system's overall performance can occur now if there is either perceptual uncertainty or mental memory workload. Perceptual uncertainty means that it is not or only with great effort possible for the user to distinguish different states or operations. Mental memory workload means that there is a great number of operations and states which are difficult to structure and the only structure that can be derived needs a high number of mental operations to be adapted to the task structure.

Some scientists concerned with model building speak of human-computer "communication", regarding communication elements also (e.g., Oberquelle, Kupka, and Maas, 1983). They regard the human user and the computer system as two communication partners and postulate:

1. Communication serves to co-ordinate (real and symbolic) actions of several agents.

2. Communication is determined by the objectives of all participants (intentions).

3. Communication depends on comparable premises for understanding (knowledge and conventions).

4. Communication can refer to the communication process itself and to its preconditions (metacommunication)

5. Communication is always coupled with expectations concerning the partner (partner model).

6. In communication there is a trend towards economical behavior.

From these statements they derive a special structure representing main components of human-computer communication, see Figure 3.

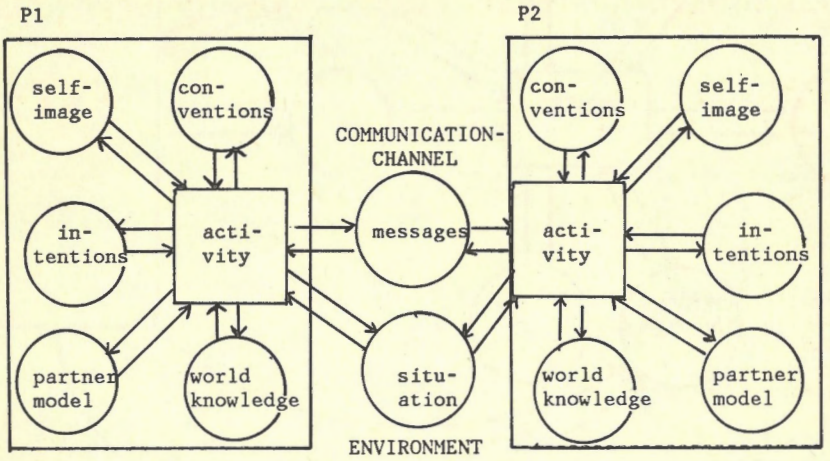


Figure 3: A model of the components of two communicating partners (from which one could be a machine)

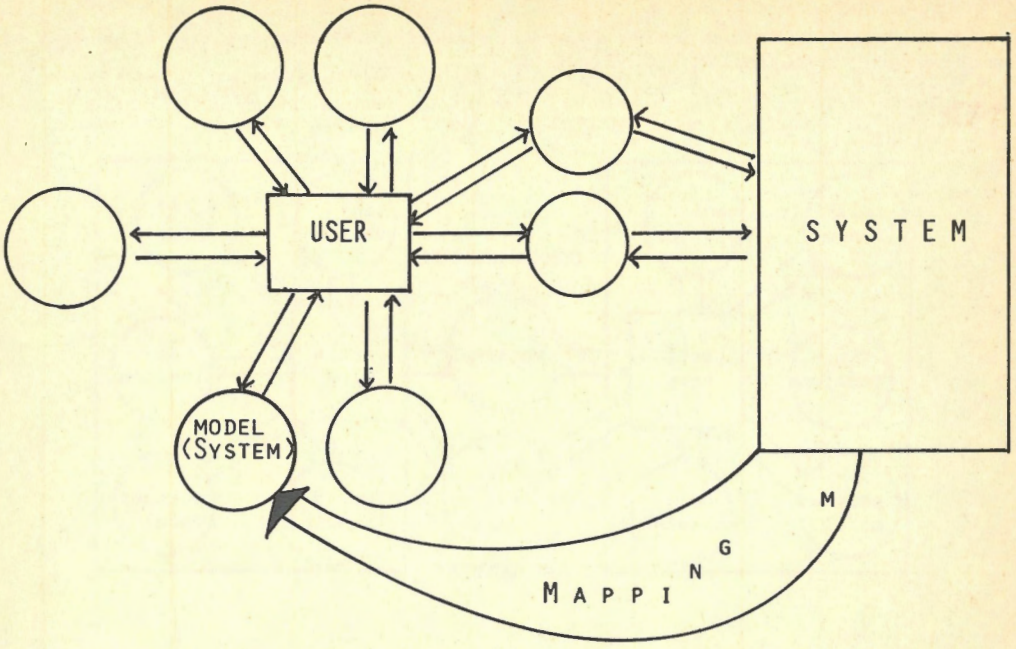
It must be noticed here, however, that in speaking of "Communication" intentional acts are more or less explicitly assumed for both communication partners, that means for the machine side also. It is questionnable whether this assumption is realistic.

2. MENTAL REPRESENTATIONS - A REPRESENTATIONAL FRAMEWORK TO DESCRIBE IT

In the component model of human-computer communication (Oberquelle, Kupka, Maas 1983) one component is called "model of the partner". This component means the mental representation the user has acquired about the system structure on the human component side. For users such a knowledge is necessary to control and predict the behavior of the system. Mental representations map world states where understanding is described by internal processes effecting the mental representation by changing states in this representation, and it therefore models a real process of changing the assigned states in the world (see Figure 4; compare Bobrow, 1975; Jagodzinski, 1983; Tauber, 1984).

This kind of understanding provides the user with the competence to describe a special process internally by effects on the mental representation and to delegate it then to the real system.

One reasonable approach to design human-computer interfaces is to consider users' mental representations of the system. It must be asked which kind and structure of a symbolic representation (interface) is most suitable for the user to build up an internal representation. Users' mental representations are internal models of an external system specified by the function to control his actions in interaction with a machine system. The main purpose for a user interacting with a machine is to perform a



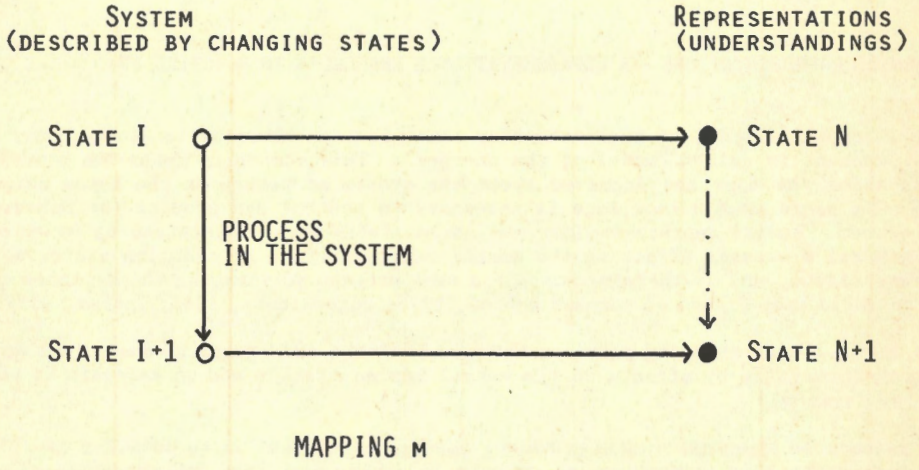


FIGURE 4: MAPPING OF THE SYSTEM ON THE USER MODEL OF THE SYSTEM

well-defined class of tasks by means of that machine. Therefore it must be assumed that the mental model the user builds up about the real system is influenced by his representation of the task structure. The architecture of the system determines how the task can be performed. The underlying "task logic" (specific for each system) is the task space.

Task spaces are defined by a set of tasks T(1), ..., T(n) changing the state of an object world, described by states S(1), ..., S(m). Each task is described by an initial state S(I) and a goal state S(G).

Tasks entail subtasks, and the relationship between tasks and subtasks defines the task structure. In the task structure (of hierarchical order) primitive tasks (not decomposable) and composed tasks (composed of primitive and composed tasks) can be distinguished (see Figure 5).

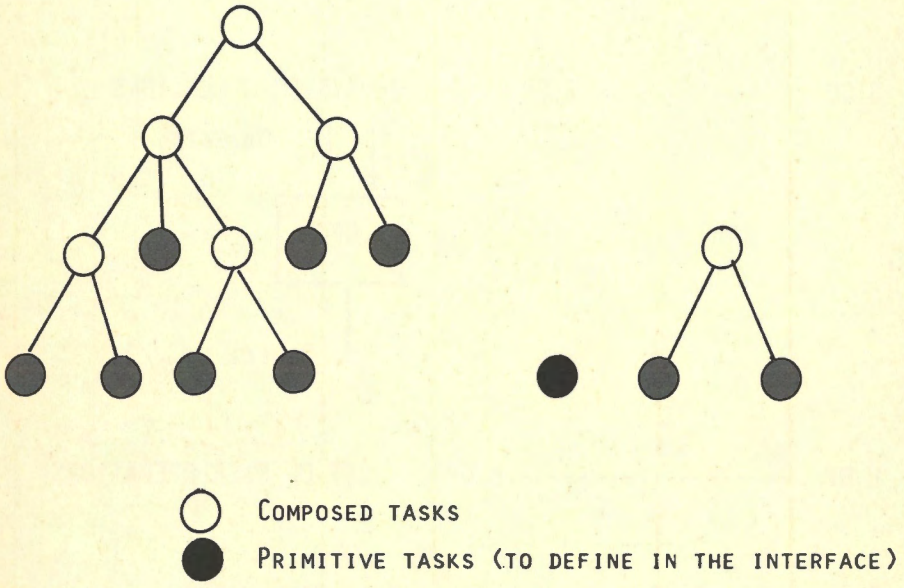


Figure 5: Example for a task-structure

Each system defines a task-space with which the user should be acquainted. Halasz and Moran (1982) and Moran (1983) showed that users' knowledge of the task-logic is sometimes quite different to the task logic defined by the system. The reason is that tasks are often performed by means of other tools. Moran (1983) defines "task space" as always to be the internally represented task space related to a system. He furthermore distinguishes between an external task space (not computer related) and an internal task space (related to a computer system).

To perform tasks by means of a system users must know how "real objects" are represented in the system (system objects), and which operations (system operations) are provided by the system to manipulate system objects and to change the state of the object world.

Each mentally represented task is described by a composition of system operations and user operations. User operations have the same effect as system functions, they manipulate objects and change the state of the object-world, but are performed by the user. In addition to the knowledge about the task-space users need a model about the functionality of the system with respect to the underlying task space for understanding the system (see Figure 6).

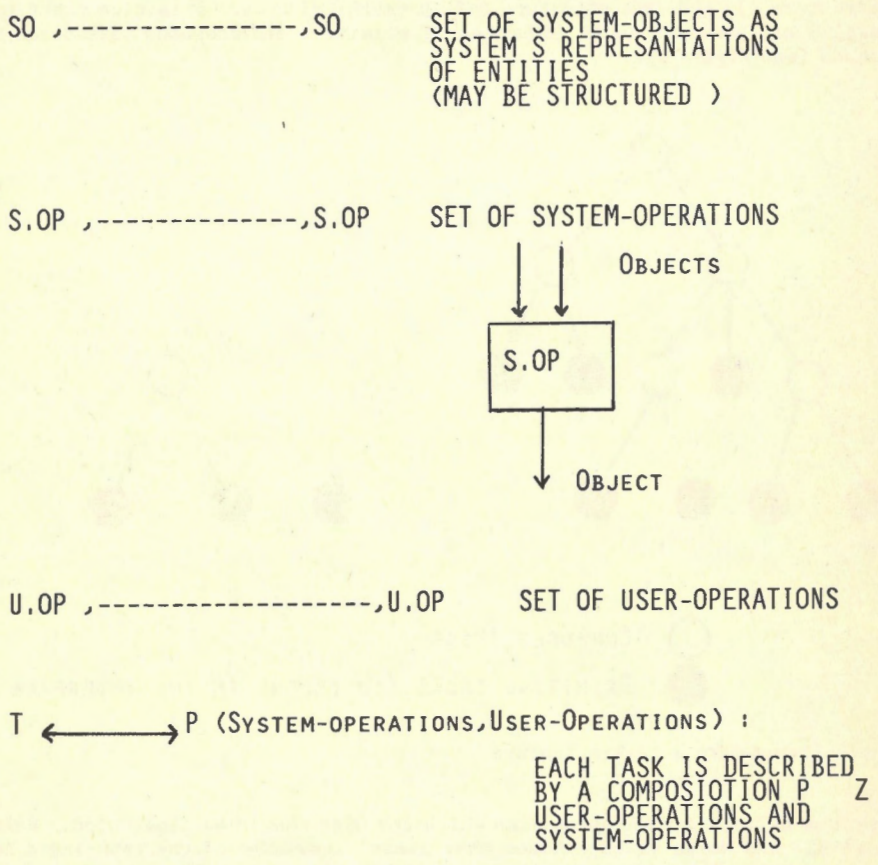


Figure 6: Semantic level (functionality)

The functionality itself must be described and evoked by a suitable language. Commands are related to a semantic procedure (composed of system operations) and grouped together in contexts (system states defined by the sets of commands applicable in the

state). Objects (as arguments) manipulated by the semantic procedure which is triggered by the commands must be named by descriptors (see Figure 7)

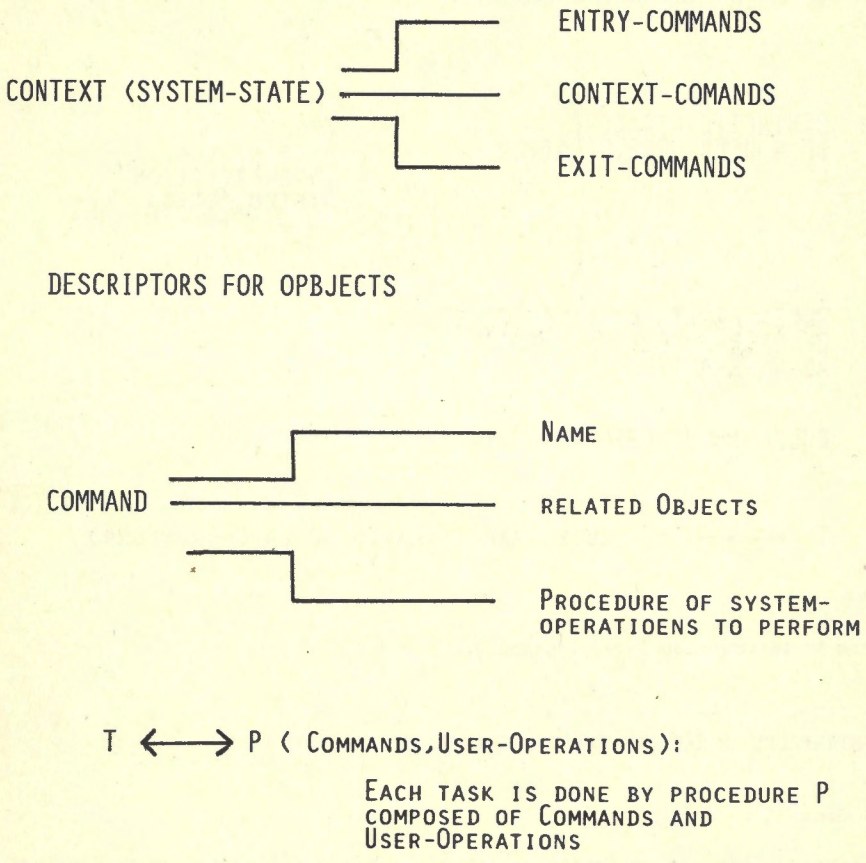


Figure 7: Syntactic level (language)

Finally, actions must be performed by the user and the system in a sequence of users' specifications and system's responses. The time sequence is determined by the command-argument structure and the command contexts (syntactic hierarchy) and is a product of user's or system's decisions during the dialogue-process. Each command must be specified by a physical action and each argument also. Users need knowledge about the process to specify a command and about the time structure of the possible specifications (see Figure 8).

This short and roughly illustrated picture of a layered mental representation of systems is based on the work of Moran (1981; CGL) which is central for HCI research.

The point of view for mental representations given in his command-language grammar (CGL) is the basis for a representational framework to describe representations of a system. The main point is the idea of layered representations, which represents the process of task performing on different levels (task space level, semantic level, syntactic level, and interaction level).

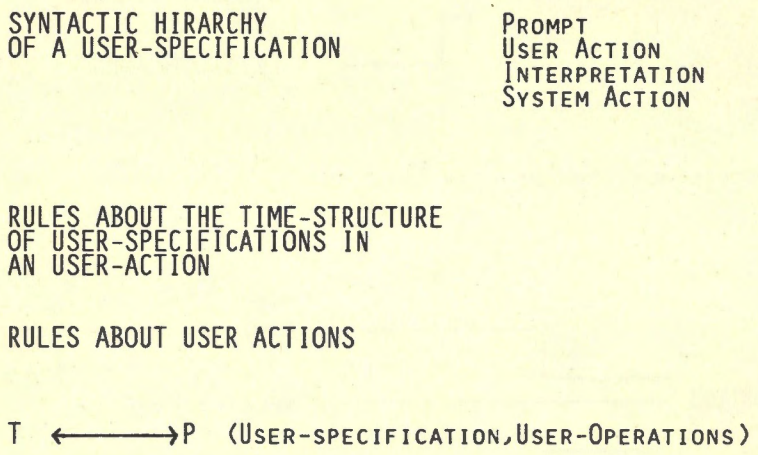


Figure 8: Interaction level (actions)

3. INTERFACES AS REPRESENTATIONS OF SYSTEMS

3.1. General remarks

The crucial point in designing an interface for a HCI system is to find out an architecture which can serve as intended mental representation for the user. This approach needs to specify a complete representation of the virtuality of a machine with respect to a chosen task-space which can serve as a "theory of the system" for the user.

The system must be described completely and exhaustively on the three levels mentioned above (semantic, syntactic, and interaction level) and reveal not only the visible components of a system but also the whole conceptual world. The definition (we think the broadest and best) of an interface as a representation of the virtuality of the machine system matches in a certain sense the definition of the conceptual interface as a set of mutual suppositions.

3.2. The conceptual model and its notation

To build up a concrete interface on the basis of a representational framework requires a modelling process. The first step is to describe the conceptual world as it should be presented to the user in a specific sign notation. For this the conceptual model de-

fined by Moran (1981) and Norman (1983) is suited best at present. The model is presented in a specific sign notation and its function is to describe a complete representation of the virtual system. The symbolic notation concerns the task-space as well as the three representation levels: semantic, syntactic, and interaction, and the elements of each level.

The most overall technique of notation has been proposed by Moran (1981) with his CLG. He uses a notation technique to develop the conceptual world straight forward and represents the semantic and the syntactic level in a frame-like notation. He starts with the semantic definition of a system:

```
X-SYSTEM = (A SYSTEM
            NAME = ' . . .  '
            ENTITIES = (SET =  . . .  )
            OPERATIONS = (SET =  . . .  ) )
```

Further he specifies each entity by

```
X-ENTITY = (AN ENTITY
               OF (Y-ENTITY)
               NAME =  . . .  )

or Z-ENTITY = (AN ENTITY
               REPRESENTS (X-OBJECT)
               NAME =  . . .  )

or Y-ENTITY = (A LIST)
```

and then he specifies each system operation by

```
   X-OPERATION = (A SYSTEM-OPERATION
                  OBJECT = (A PARAMETER)
                  VALUE = (A X-ENTITY)
                  DEFAULT VALUE = ... ) )
```

Finally for each task to perform a procedure built from system operations and user operations can be specified:

```
   X-METHOD = (A SEMANTIC-METHOD
                 FOR X-TASK

              DO (... Description of the procedure by means of
                  a control flow through the used system's and
                  user's operations ))
```

X, Y, Z in connection with entities, tasks, and operations mean a concrete entity, task, or operation.

The same technique to build up a concept hierarchy is also used for the syntactic level to specify command contexts (describing the system states), to name entities by descriptors, to describe single commands by means of the related semantic procedure, and to define a task by a procedure of commands and user's operations.

Quite different is the technique for describing the interaction level. To each command-argument specification an interaction tree must be assigned describing the time sequence of the single user's and the system's actions.

Beyond the description of a single command specification rules consistent with respect to the user are needed which describe similar ways of naming commands and arguments, and evoking them by physical actions. Here some work has been done by means of the notation of productions (Moran 1981) or higher-ordered rules (Payne and Green, 1983).

Most methods to notate a conceptual model are knowledge based and provide a complete description of the system's representation.

Jacob (1983a, 1983b) developed a specification technique which specifies the semantic level by means of LISP functions and the syntactic as well as the interaction level by means of special kinds of state-transition diagrams. His symbolic descriptions serves as a model for prototypes of interfaces. He does not specify, however, the conceptual world underlying the interface like the CLG technique. Conceptual classes (chunking), e.g., or rules for consistency are not specified.

4. DECISIONS IN THE DESIGN PROCESS

4.1 Decision steps

The method of modelling interfaces by means of conceptual models indicates how many decisions a designer has to take on the construction of an interface which he wants to base on an explicit user model. Table 1 illustrates this point.

In a top-down designing process, beginning with a given task space and proceeding to the interface level, a designer has to decide which model he will choose on each level (see Figure 9).

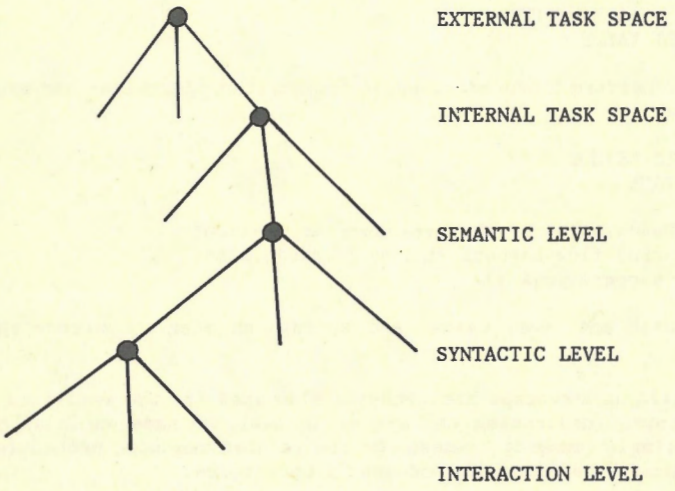


Figure 9: The model decision tree of the designer

By means of these models selected the designer builds up a complete interface which has the following user perceived qualities: flexibility (the system provides one or more kinds of interfaces), self explanation (the system explicits the conceptual model), consistency (there are rules for chunking single concepts), etc.

An important point for the decisions on each level is that an interface should provide reduction of complexity for the user as can be seen from the questions in table 1.

TABLE 1

LEVEL	THINGS TO DECIDE	SPECIAL QUESTIONS
Task-Space	- Definition of the objects - Object structure - Primitive tasks - Composed tasks	- How primitive are primitives? - How complex is the task-structure? - Are objects chunked and how?
Semantic-Level	- System objects as representations of objects - System operations - User operations	- How many systems operations are to define? - How complex is the functionality of each system operation? - How complex are the procedures made from system and user operations assigned to each primitive task? - How complex are user operations?
Syntactic-Level	- Commands and related semantic procedure - Contexts as chunking of commands to system states - Description of objects	- How complex is the semantic procedure to a command; should a command be related to a semantic system operation, to a procedure of system operations or to a complete task? - What commands should be collected to a context? - How complex is a context? - Are there defaults?
Interaction-Level	- Rules to specify a user action - Definition of all single user actions - Visible systems actions	- How complex are the operations on the keyboard, with the mouse, ... - Are there similar or equivalent techniques to specify different actions (consistency)

4.2 Reducing complexity

Complex systems are defined by having a great number of states, and transitions between states and operations. Hereby it must be noticed that complexity can only be regarded with respect to the user of a system, and consequently it only can be defined for a given representation of a system and not for the system itself (Rasmussen and Lind, 1981). Hence we must regard mainly the user's mental representations of the system structure in relation to the symbolic representation of the system, the human-machine interface, i.e. the organisation of the single displays, their reference/relation to each other, and the relation of the symbols to the causal structure of the system. (see here and for the following Figure 2)

Complexity on the representation side is created with respect to the human user if there is a great amount of elements and relations between these elements which have to

be memorized by the user. Reducing complexity at this level (the interaction level after Moran 1981) means to display most of the information needed at the screen: (1) symbolizing states, their transitions, and organization by an interpretable screen layout; (2) symbolizing chunking of operations which evoke an association of the underlying causal structure. Some investigations on this point have been done by Rohr and Keppel (1984, 1985) and Keppel and Rohr (1984).

Another point influencing complexity at this level is perceptual uncertainty. If two different states allowing only different kinds of operations cannot be distinguished from each other by the user, the user has to memorize operational sequences in the time history for to know which state he has reached.

Complexity can occur in higher levels also. First the represented task structure itself can be complex. Furthermore, even if the represented system structure as well as the task structure are not complex, the transformation of the system structure to a virtual machine (in the sense defined in 2.3) by means of the task structure (external task space after Moran 1981) can require a great number of transformation rules which makes the complete represented task space very complex.

Last not least, if chunking strategies (i.e. categorial classifications of command operations) implemented in the interface (symbolic representation of the system) do not correspond with mental chunking strategies the user must again develop categorization rules affecting mental workload, and enlarging mental complexity.

Reducing complexity in the last two points means to analyze mental task spaces and categorization rules of the human user, and consequently to adapt the system's symbolic representation to them.

5. MODELS IN HUMAN-COMPUTER INTERACTION - A SYNOPSIS

As defined above (1.1.), models of human-computer interaction can be characterized by the substrate chosen, the purpose and subject they serve, the components of the object chosen, and the quality in predicting or describing the behavior, performance, or structure of the object.

Most models we try to characterize in the following, have either the purpose to gain design criteria for a user interface architecture or to teach people better to use a computer system. Symbolic sign notations are taken as model substrates, i.e., formal languages or graphical sign notations.

The objects mainly chosen are:

1. The machine, defined by components which describe interfaces for control operations of the user and/or designer

2. The human user, defined by components of cognitive processes which match requirements of control operations at the machine

3. The human-computer interface as representation of the virtuality of the machine, defined by components of tasks structure, human mental representations and processes, and machine system (symbolic representation)

4. The overall human-computer communication system, selecting components which allow a structural description of the system.

Table 2

	SUBSTRATE	PURPOSE	OBJECT	QUALITY NOTE
CONCEPTUAL MODEL (Norman 1983)	formal language	find design criteria	(3)	describes structure, not tested out
IFIP MODEL (Dzida 1983)	graphical sign notation	find design criteria	(1)	only possible to say where points of modelling are, no testing possible in that form
SITES-MODES-TRAILS MODEL (Nievergelt 1983)	quasi formal language and graphical sign notation	find design criteria	(2)	describes structure, not tested out
USER-AGENT MODEL	graphical sign notation	find design criteria (intel-ligent interface	(3)	describes components for modelling, testing not possible in that form
KEY-STROKE MODEL (Card,Moran,Newell 1983)	formal/ mathematics	find design criteria	(3) part-ially	describes and predicts behavior
COMPONENT MODEL OF COMMUNICATION (Oberquelle,Kupka, Maas 1983)	graphical sign notation	? mainly component selection	(4)	describes components of communication, no testing possible in that form
MODELS OF MENTAL REPRESANTATIONS AND PROCESSES (general)	formal language, mathematics, graphical sign notation	explaining user behavior	(2)	several models tested out can applied to model HCI systems
PROTOTYPES (general)	programming language	test design criteria	(1)	describes software structures

The quality of prediction or description of the current models is difficult to evaluate in most of the cases. The main raison is that there have been only few attempts yet to test them out.

Table 2 shows some current models used in describing human-computer interaction, and their classification by means of the characteristics defined above.

REFERENCES

Bobrow, D.G. (1975). Dimensions of representation. In: Bobrow, D.G., Collins, A. (Eds.). Representation and Understanding, Academic Press, New-York.

Dzida, W. (1983). Das IFIP-Modell zur Benutzerschnittstellen. Office Management.

Klaus, G., Liebscher, H. (1979). Lexikon der Kybernetik, Fisher, Frankfurt/M.

Gunzenhaeuser, R. (1984). Lernen als Dimension der Mensch- Maschine- Kommunikation. In: Schauer, H., Tauber, M.J. (Eds.). Psychologie der Computerbenutzung, Wien-Muenchen.

Halasz, F.G., Moran, T.P. (1982). Analogy considered as harmful. In: Moran, T.P. (Ed.). Eight short papers on user psychology. Palo Alto, Xerox PARC, pp. 33-36.

Jacob, R.J.K. (1983a). Using formal specifications in the design of a Human- Computer-Interface. Comm. ACM, 26, pp. 259-264.

Jacob, R.J.K. (1983b). Executable specifications for a Human- Computer- Interface. CHI'83 Proceedings, SIGCHI, ACM.

Jagodzinski, A.P. (1983). A theoretical basis for the representation of on-line computer systems to naive users. Int. Journal of Man-Machine Studies 18, pp. 215-252.

Johannsen, G., Boller, H.E., Douges, E., Stein, W. (1977). Der Mensch im Regelkreis, R. Oldenburg, Muenchen.

Keppel, E., Rohr, G. (1984). Prototyping - A method to explore human factor aspects in application software. First International Symposium on Human Factors in Organizational Design and Management, Hawai.

Moran, T.P. (1981). The command language grammar: a representation for the user interface of interactive computer systems. Int. Journal of Man-Machine Studies, 15, pp. 3-50.

Moran, T.P. (1983). Getting into a System: External- Internal Task Mapping Analysis. CHI'83 Proceedings, SIGCHI, ACM.

Nievergelt, J. (1983). Die Gestaltung der Mensch- Machine- Schnittstelle. In: Schauer, H., Tauber, M.J. (Eds.). Psychologie des Programmierens. Oldenburg, Wien-Muenchen.

Norman, D.A. (1983). Some Observations on mental models. In: Gentner, D., Stevens, A.L., Mental Models, Erlbaum Ass., Hillsdale N.J.

Oberquelle, H., Kupka, I., Maass, S. (1983). A view of Human- Machine Communication and Co-operation. Int. Journal of Man-Machine-Studies, 19, 4, pp. 309-333.

Payne, S.J., Green, T.R.G. (1983). The user's perception of the interaction language: a two-level model. CHI'83 Proceedings, SIGCHI, ACM.

Rasmussen, J. and Lind, M. (1981). Coping with complexity. In: Stassen, H.G. and Thijis, W. (Eds.), Proceedings of the First European Conference on Human Decision Making and Manual Control (Delft University of Technology, Delft)

Rohr, G., Keppel, E. (1984). Iconic interfaces: where to use and how to construct. First International Symposium on Human Factors in Organizational Design and Management, Hawai.

Rohr, G., Keppel, E. (1985). Was sagt ein Bild? - Zur begrifflichen und bildlichen Kodierung komplexer Vorgaenge bei der Mensch-Computer- Interaktion. Bericht ueber den 34. Kongress der DGfP 1984. Hogrefe, Goettingen.

Rouse, W.B. (1980). System engineering models of human- machine interaction. North-Holland, Oxford.

Tauber, M.J. (1984). Zur Spezifikation und Konstruktion von Help-Systemen. Notizen zu Interaktiven Systemen 12, pp. 71-87.
Zemanek, H. (1982). Festvortrag 25 Jahre IFAC. In: Mitteilungsblaetter der oesterreichischen Computergesellschaft.

ON MODELS AND MODELLING IN HUMAN-COMPUTER CO-OPERATION

Horst Oberquelle

Universität Hamburg, Fachbereich Informatik
Schlüterstr.70, D-2000 Hamburg 13
Germany

> System designers take note. Design the system for the person, not for the computer, not even for yourself. ... Provide the user with an explicit model.'
>
> D.A.Norman (Norman, 1981)

In the recent literature on human-computer communication (HCC) 'models' have become a central notion. State/transition models of interactive systems are discussed again and again (from Parnas (1969) to Jacob (1983)), the sites/modes/trails model of Nievergelt is praised as a solution for problems many users encounter (Nievergelt, 1983), the model human processor is designed to analyse user behaviour (Card, Moran & Newell, 1983), an abstract model of communication between two partners has been applied to HCC (Oberquelle, Kupka & Maaß, 1983). There are lots of other papers discussing conceptual models or mental models of interactive systems, user models or the role of metaphors (see e.g. Carroll & Mack, 1982).
Obviously, humans need and use models of their 'relevant system' to be able to plan their actions and to control the obtained effects. As soon as several persons cooperate their communication will be based on their respective models. Part of their communication will serve to develop, to explain and understand models. To communicate about models presupposes that models can be represented externally and that the means of representation are known to the communicating parties. Communication about modelling tools might be necessary in addition. All kinds of modelling, formal models included, are based on the human ability to communicate in the informal mode inherent in natural language (cf. Naur, 1982).

Users of computerized systems have great difficulties in acquiring adequate models of the systems they are forced to work with. The development of systems for HCC is not just the creation of programs according to some specification, but the design of working situations for humans. That's why principles of both, hardware and cognitive ergonomics, should be applied.
One basic principle is controllability (Troy, 1981) which means that the working environment must be transparent and predictable and that the worker must be able to influence it. Models of the working environment and modelling procedures can serve exactly for this purpose.
Researchers in cognitive ergonomics are asked to improve the situation by looking for suitable models, modelling tools and modelling procedures. But to date their notions of model, relations between partial models and modelled aspects as well as the purposes of models have not been discussed in a systematic way.
A suitable model of the system relevant for HCC research may help to evaluate different proposals, to improve scientific communication and to give hints for further research activities. A uniform representation technique for different but related models might improve communication between researchers, designers and users.

Before going into the details of any specific model or representation technique the following questions shall be answered (section 1):

- What do we mean by 'model' ?
- Which aspects of 'the relevant system' are covered by models ?
- What is the role of 'modelling tools' ?

In section 2 we show how the focus of interest in HCC has been extended in recent years and illustrate this by means of nets. Today three major roles (as abstractions of agents with similar aims) influence human-computer interfaces: designer, dialog system and user. For the researcher in HCC they determine 'the relevant system', which can be represented by a meta-model.
This model is taken as a framework according to which we discuss and classify some models from the literature (section 3).
In view of the problems users have with systems and models predefined by others we ask: "When, how, by whom and for whom are models developed and transferred ?" The meta-model shows new ways of improving the modelling process. This will be discussed in section 4. Section 5 indicates some open problems and gives directions for further research.

1. MODELS, MODELLING PROCEDURES AND MODELLING TOOLS

The purpose of this section is to provide a common understanding of the notions model, modelling procedure and modelling tool and their mutual relations. Models and their characteristics are discussed first.

Several notions of 'model' are in use. Although they are all related we firstly want to point out which of them is considered:

(i) a model of an axiom system,
(ii) an abstract description of the relevant aspects of a system,
(iii) a known system with structure and behaviour analogous to the system under consideration, or
(iv) a prototypical system in the sense of 'model farm'.

Models in the sense of (iii) may better be called exact analogies. They are a special case of a metaphor. Models for HCC in the sense of (iv) do not yet exist. They should be a goal of HCC research. Even a widely accepted 'model operating system' (UNIX) does not have a 'model user interface' (Norman, 1981).
Since users and designers are primarily interested in using models to reduce the complexity of real systems the second interpretation is the most appropriate one. We can define it more precisely as follows:

A model is a communicable description
- of a certain aspect (the view)
- of a section of reality (the system)
- at some level (of abstraction or detail)
- as perceived by a human being (model builder)
- which is to serve the purposes of its users .

In terms of mathematics, a model is a homomorphic image. It serves one basic purpose: it reduces the complexity of reality by dividing it into the 'system' and the irrelevant rest and by abstracting from certain details of the system (view, level of abstraction). It allows its user to classify and explain phenomena of the modelled system. For example, if the computer user has an adequate model of his dialog system he can plan his actions and anticipate their effects; the system's behaviour will be without surprise for him. Furthermore, he is relieved of learning lots of details, since the model enables him to reconstruct the details mentally.
In two ways the boarder line of the modelled system and the chosen view are important for the model user:

1. they restrict the action and thinking space excluding possibilities not taken to be relevant by the model builder;
2. they show the wealth of possibilities provided inside the model.

As long as model builders and model users are identical or belong to the same group (e.g. edp experts) restrictions set by the model do no harm.

Normally, several models of the same system are in use simultaneously; according to different interests they describe different aspects or the same aspect at different levels of detail. This situation is illustrated in Fig.1. To understand a system from partial models it is of utmost importance that they can easily be related.

Two views of systems are often reflected in separate models:

- the composition of the system out of simpler functional units (static, spatial structure),
- the behaviour of the system as a process or process schema (dynamic, temporal structure).

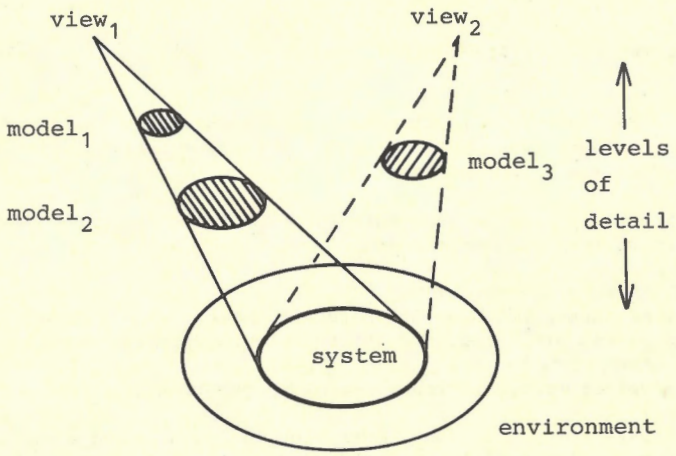


Fig. 1: Partial models of the same system

We do not want models to be formal in general since not all model users need or even understand formal models. But we demand that models are - in principle - communicable, since adequate models are often built and adjusted in the course of communication.
Depending on the circumstances, models may be descriptive as well as prescriptive. This difference of intentions helps, for example, to distinguish the above mentioned interpretations (iii) and (iv).
The notion of 'model' as explained before does not presuppose that the modelled system exists before the model is built. For example, usually the designer of an interactive system creates a model in form of a specification which is of prescriptive nature for the implementor(s) of that system. After a correct implementation the specification can serve as a descriptive model for the user of the system - if he or she is able to understand the designer's model at all.

Sometimes the internal organization and processes of a system or a component are not known in detail. For a coarse description which allows to explain the system's behaviour sufficiently well it is not clear, whether it is a model according to our definition or a suitable abstract analogy. Nevertheless, we call descriptions of this type models, too.

Modelling is the process of forming mental models of a special system or of a class of systems for a special purpose.

This can mean three different things:

(a) the construction of a model by abstracting from a given system during its usage;

(b) learning about models built by others (including the transfer of models via computers);

(c) creating a model prior to building a system based on experience gained from (a) and (b).

Learning a model built by someone else may evoke a special problem if the intentions of model builder and model user are too different. In general, models are adapted to changing needs during their usage - forbidding a sharp distinction between a modelling phase and a phase of model application. Updating or refreshing a model by its forgetful user is a special case of adaptation.

The external representation of models is an auxiliary step for forming mental models or for the construction of artificial systems. External representations make evident whether one has got a clear understanding of the system under consideration.

Modelling tools are special description tools used for the external representation of models in addition to natural language.

Modelling tools may be graphics or formal systems. To understand an externally represented model one has to understand the modelling tools and the metaphors behind them. Inappropriate tools may provide a bias in one's understanding.
The learning of modelling tools is easier if the tools are based on few basic concepts and can be applied to many situations, especially to situations well-known to the learner.
To have good modelling tools does not necessarily mean to use formal tools in the sense of mathematics. Carefully chosen graphics combined with explanations based on a precise and consistent terminology may be appropriate for certain groups of model users, for instance for computer novices and their managers. Net representations we are going to use belong to this kind of modelling tools.

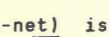
2. A META-MODEL OF HUMAN-COMPUTER CO-OPERATION

In order to improve the actual discussion on different aspects of HCC and as a framework in which to search for new solutions for problems of HCC we want to develop a model of the system which determines HCC. Relations to former views of HCC are indicated. One can easily see that since the beginning of HCC the field of interest has been extended in three dimensions:

- in space: adding new system components;
- in time: considering system creation and modification in addition to plain system usage;
- in aspects: taking notice of pragmatic factors in addition to syntax and semantics.

At the beginning of HCC machine language programmers were the only users of computers. The 'system' they were using was an automaton which they knew in every detail. On a very abstract level its structure can be modelled as a net of channels and agencies as done in Fig.2 *).

---

*) A net of channels and agencies (CA-net) is a directed graph whose nodes represent functional units (○ for channels, □ for agencies) and whose arcs represent the access rights of the agencies ( read/get from C, write/put to C). Only nodes of different types may be connected. Isolated nodes are forbidden.

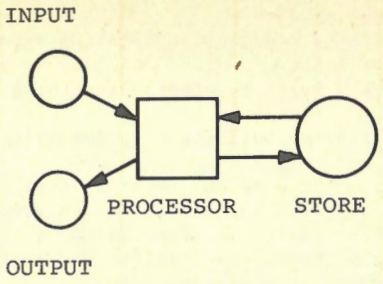


Fig. 2: The computer as an automaton
(CA-net representation)

During the development of interactive systems virtual machines*) have been designed which can be interpreted as refinements of the model in Fig.2. A design which was used for an early experimental system for interactive programming with a dialog-specific frame and exchangeable kernels for different applications (RDS) (Kupka, Oberquelle & Wilsing, 1975) is shown in Fig.3. The model primarily served as a basis for the precise definition of syntax and semantics of the dialog language. The separation of concrete input and output from abstract input and output was provided for experimentation with semantically equivalent language constructs. The syntax of inputs is state-dependent.

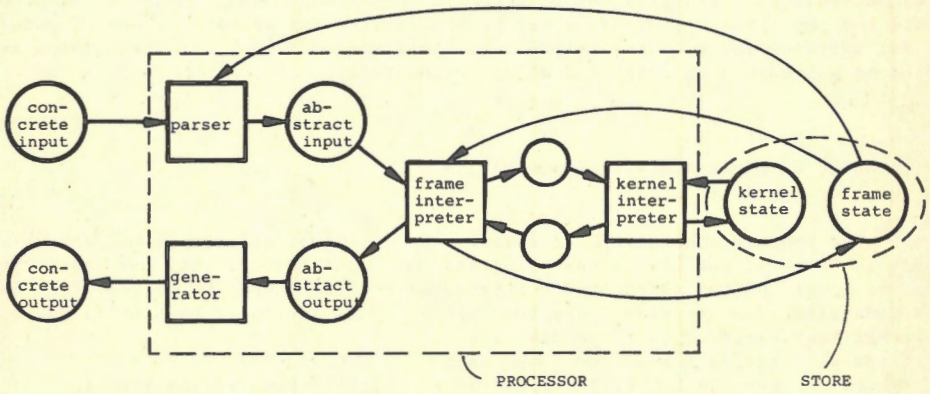


Fig. 3: Structure of the RDS system
(CA-net representation)

When pragmatic aspects were taken into account it became clear that a larger system had to be considered. This was reflected in the concept of an (abstract) dialog processing system (see Fig.4) which is a model of a system with two active agents (dialog initiator and dialog responder) both of which can represent a human or a machine.
This static model can be 'unfolded' into a means/activity net**) (see Fig.5) which can be used to introduce a general terminology for dynamic aspects of human-

---

*) Virtual machines describe the functionality of a system in terms of abstract functional units and their behaviour. Implementation details and details of the underlying hardware are suppressed.

**) In a means/activity net (MA-net) the flow of means (◯) through activities (□) is represented using the same graphic elements and conventions as in CA-nets.

computer dialogs (for details see Oberquelle (1980a)). For example, the concept of a 'dialog step' can be easily explained (see Fig.5). The same model underlies the discussion of dialog styles, e.g. user initiated vs. system-initiated dialog steps, by Dehning, Essig & Maaß, (1981)).

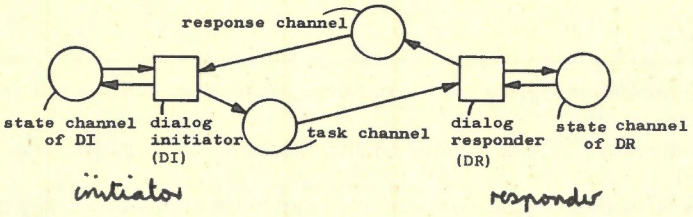


Fig. 4: Dialog processing system
(CA-net representation)

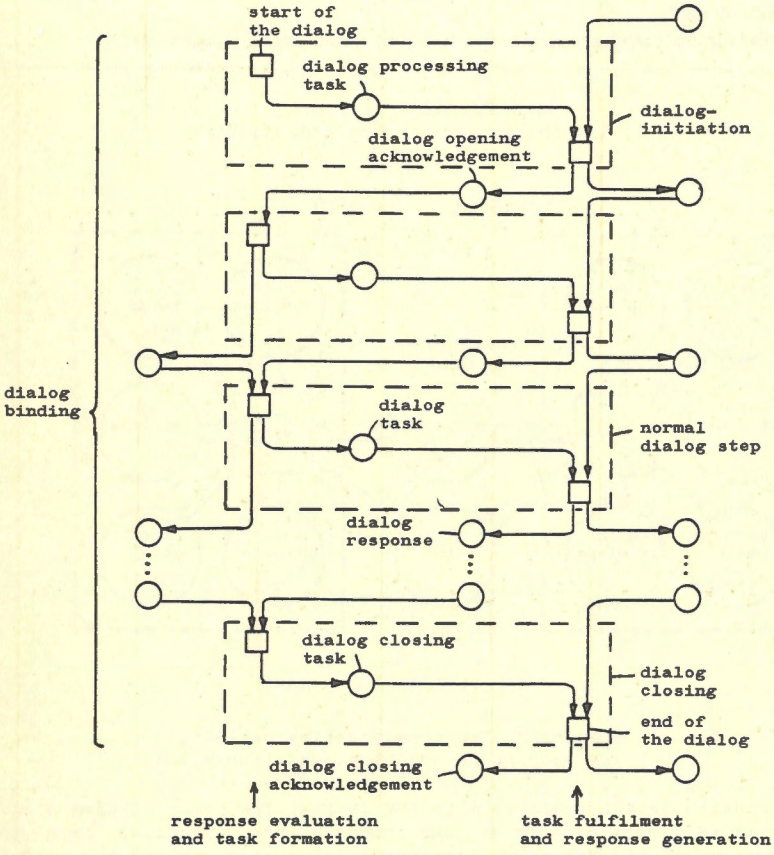


Fig. 5: The dialog as a process
(MA-net representation)

In order to improve the situation of (edp-)naive users another situation was analyzed: human-human communication (HHC). One aim was to see whether HCC was restricted in comparison with HHC and in what regard. The other was to look for new ideas for the design of interactive systems. Both questions were discussed on the basis of a model of communication between two abstract partners. The model reflects six pragmatic characteristics of communication applicable to HHC and HCC (for details see Oberquelle, Kupka, & Maaß, (1983)).

(1) Communication serves to co-ordinate (real and symbolic) actions of several agents.
(2) Communication is determined by the objectives of all participants (intentions).
(3) Communication depends on comparable premises for understanding (knowledge and conventions).
(4) Communication can refer to the communication process itself and to its preconditions (metacommunication).
(5) Communication is always coupled with expectations concerning the partner (partner model).
(6) In communication there is a trend towards economical behaviour.

Fig. 6: Six characteristics of communication

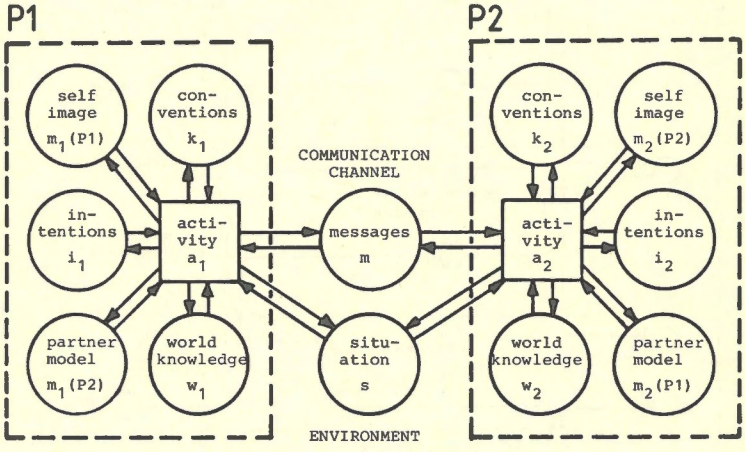


Fig. 7: A model of two communicating partners
(CA-net and MA-net representation mixed)

The main restriction in HCC compared with HHC is that the communicating behaviour of the computer is of algorithmic nature and that all its reactions have to be planned in advance by its designer. For some users dialog systems may seem to be independent communication partners, especially if users don't know how and by whom they have been designed and programmed. If an interactive system is composed of several components the combined intentions of their designers influence the human-machine interface. Users may interpret them as the intention of the dialog system.

Although dialog systems tend to become independent with own intentions (virtual independence, virtual intentions (for details see Oberquelle, Kupka & Maaß (1983)), it must be emphazised that the communicating behaviour of a dialog system is caused by its designers through delegation. To communicate successfully with an invariable system the human user has to adapt himself to its algorithmic communicating behaviour. He can only modify it within the limits set by the designer.

The communication model shows that two kinds of models play an important role in communication: self images and partner models. In interpersonal communication these models are partially built up and adjusted in meta-communication. Co-operative partners help each other to develop adequate partner models.

Dialog systems, as well, can be thought of as having partner models and self images: they adapt their behaviour according to stored user models and explain their own facilities in reaction to HELP commands on the basis of a model of the system. The realization that these models are created and put into the machine by the designer leads to a further extension of the relevant system adding the designer as a third component. The long-term processes running in this system are called human-computer co-operation, including learning about modelling tools and models (e.g. from documentation), delegation and HCC.

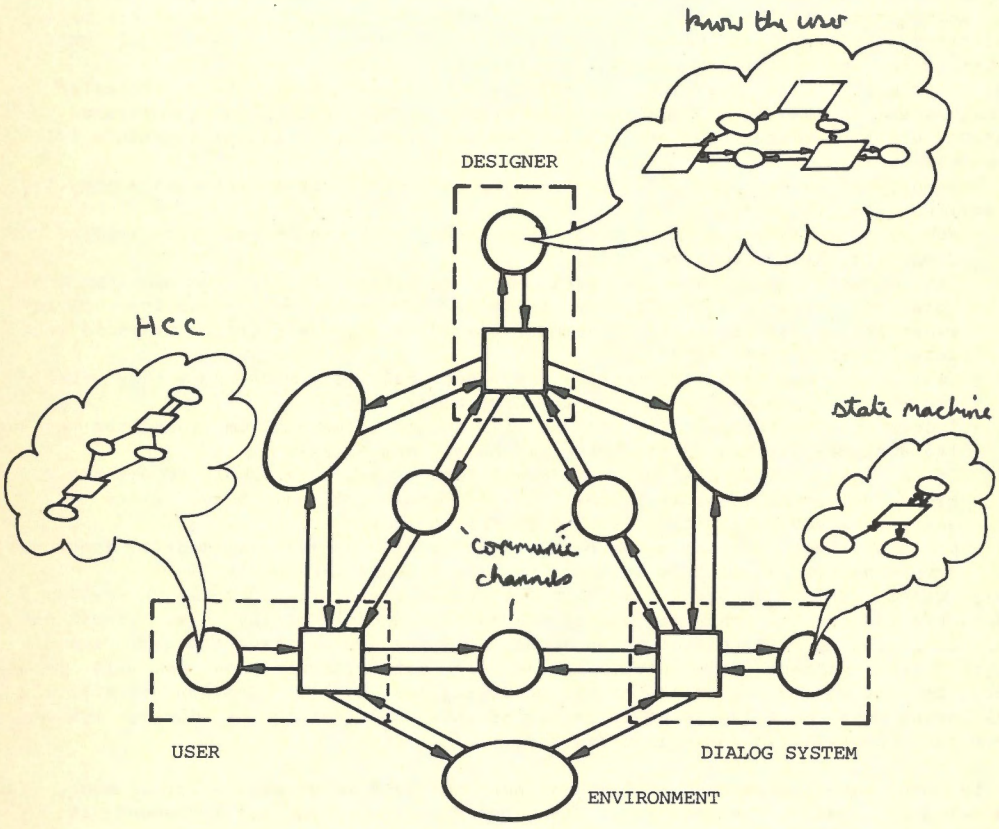


Fig. 8: The meta-model of human-computer co-operation (with an indication of different structural models of the 'system')

The states of the three agencies of this meta-model can be refined according to the communication model. The partner models and self images must be replaced by models of those parts of the system known to the agencies and considered relevant for the agencies' tasks. Fig.8 indicates quite different models which the agencies may have of the whole system. The presented meta-model will serve as a background for the following discussions.

3. ROLES AND EXISTING MODELS

The three roles of the meta-model, designer, user and dialog system, can be characterized by their special interests in models. Models useful for the different roles may differ in the degree of formality. After a presentation of role-specific interests and restrictions some models described in the literature on HCC will be evaluated.

The typical users of dialog systems (application expert, edp-naive) are interested in a manageable model of a sufficiently large portion of the organization they are working in. This must include a model of the dialog system, of its user interface and of other interfaces of the dialog system through which the user can influence his environment or is influenced himself. These additional interfaces are mainly application-dependent. The model can be described in terms of functional units, states, means and activities relevant for the users' tasks.
Due to their little training in formal methods users will not be interested in formal models as long as informal or semi-formal models allow to understand the system. Graphic representations usually are preferred to strings ("1 picture tells more than 1000 words.").
At present, users of interactive systems are in a difficult position when they try to develop adequate models:

- Models or metaphors used in their traditional work cannot easily be applied to computers (Halasz & Moran, 1982).
- The objects they have to deal with disappear behind the human-machine interface. The possibility to build a suitable model during direct manipulation of objects is almost lost. The paperless office tries to make this 'facilitation' perfect.
- Taking courses in informatics does not help very much since they usually present the computer as seen by the programmer.
- System documentations are crammed with details. They seldom reveal the model the designer has had in his mind - if he had one at all.
  If a user has to deal with different subsystems it is nearly impossible for him to develop a consistent model which covers all of them, since their designers certainly will have used different models.
- Left on their own, users are hardly able to develop suitable models. Even for simple pocket calculators it seems to be difficult (Young, 1981).

Today users can be content if explicit models are provided at all. However, if models are too complex, they may exceed the memory capacity of the user. This can lead to increased meta-communication about the model and distract the user from his original task. Without such possibilities for meta-communication he will feel uncertain and discontent, since he has partially lost control. The idea of actively influencing model building in early phases of the development of dialog systems seems to be unrealistic for most users today.

Up to now, the primary task of the designers has been to produce a formal model of the planned dialog system, i.e. its specification, and to implement it by programming and delegation.
One main difficulty is to bridge the gap between the vague knowledge about the future users and about the necessary functionality on the one hand and the formal specification on the other. The same kind of difficulty arises when the designers try to explain the finished system to users - be it by written material or so-called self-explaining systems.

They seem to be in a situation similar to that of an architect who is able to make precise plans for brick-layers, electricians or plumbers, but has little means to explain his design and its functionality to customers.
Semi-formal methods established to improve user-designer communication seem to be necessary.

The dialog system introduced as the third role in the meta-model (Fig.8) has no interests of its own. It may contain models used for purposes set by the designer. Each dialog system contains at least implicitly the partner model and self image of its designer.
Explicit models inside the machine used to control its communicating behaviour must be algorithmic ones. Models used for explanations only must be algorithmic as well, but they can use informal descriptions represented by text and graphics. All explicit models are invariable after their delegation.

Models described in the literature are usually developed and used by designers. We present a selection of them and discuss their range. Some of them are of interest for users, as well.

The oldest class of models are finite state machines which are process models of the dialog system (Parnas, 1969). They are created by designers to show all state-dependent possibilities users have for their inputs. In refined versions these models are supplemented with descriptions of system reactions or effects which inputs have on the stored data.
Such models are used on different levels of detail and formality ranging from rough representations for teaching purposes (see Fig.10) to totally formal representations seeming more useful for the designer than for the user (see e.g. Jacob (1983)).
The complexity of state/transition diagrams can be reduced, for example, by hierarchical decomposition as done by Denert (1977) or by showing the correspondence between sets of states and functional units which they belong to as done in Fig.9 and Fig.10.

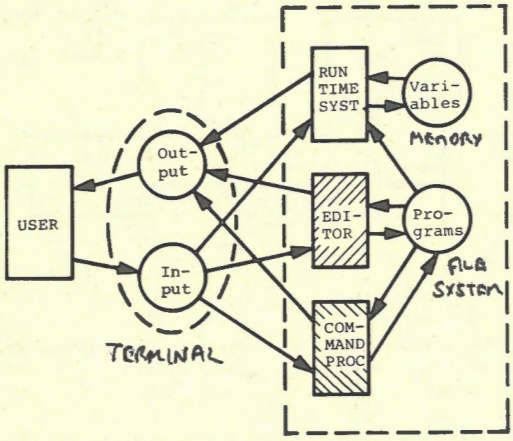


Fig. 9: Virtual machine for a Pascal programmer

Models as in Fig.9 may be used to give the user an impression of the architecture of the virtual machine he is communicating with (but only if the different components are relevant for the user's tasks).

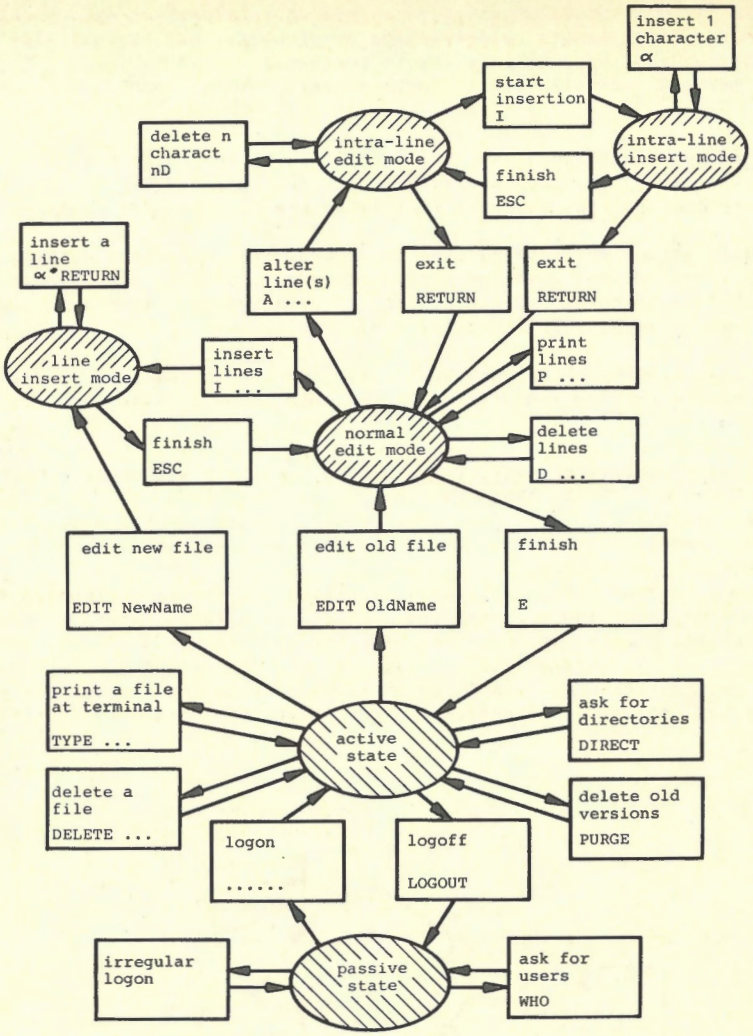


Fig. 10: State/transition net for a Pascal programmer (section, following Kupka (1982))

The IFIP model of interactive systems (Dzida, 1983) describes the static structure of a system larger than it is covered by state machine models. With respect to the meta-model it is the subsystem consisting of a user, the dialog system and their organizational environment. System development and modification are not explicitly covered.
The IFIP model is intended to give a standard refinement of the computerized part and to allow an explanation of actually discussed concepts as partial abstractions of it. For example, virtual terminals and interactive tools are one view (Fig.11), application-independent dialog systems are another. See Fig.12 for a representation

of the RDS system of Fig.3 in this view. Further refinements of the machine part may serve as a guideline for modularization and implementation. Refinements of the organizational environment may help the user to get a better understanding of his entire working situation.

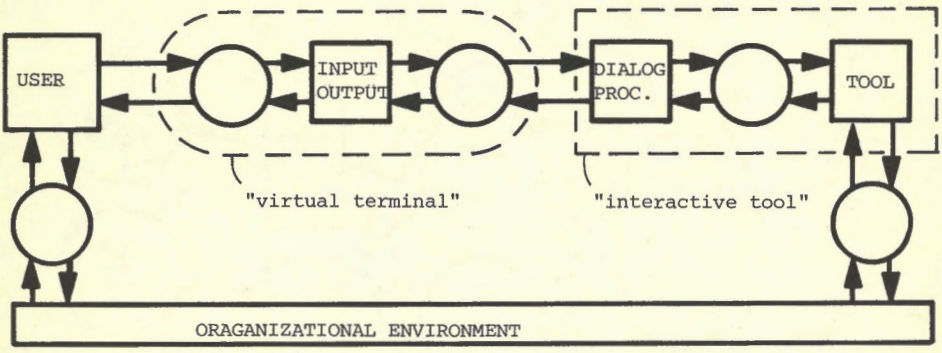


Fig. 11: The IFIP model of interactive systems
(with 'virtual terminal' and 'interactive tool')

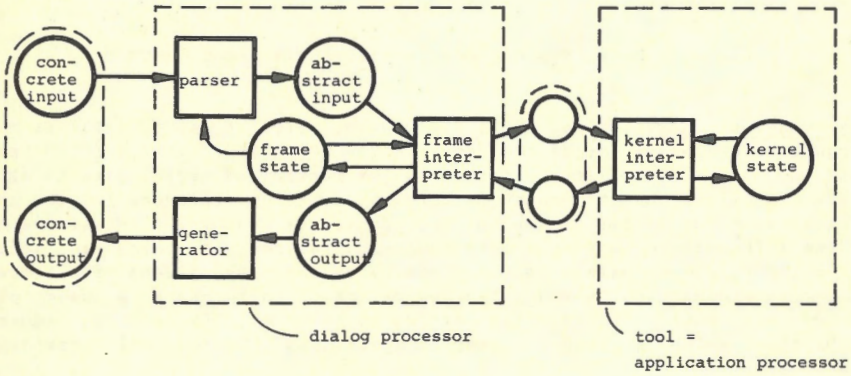


Fig. 12: The RDS system as an 'applications-independent dialog system' (frame) using a 'tool' (kernel)

The Model Human Processor (Card, Moran & Newell, 1983) is a very abstract structural model of users (see Fig.13). The associated activities are low-level. It allows designers to analyse a user's information processing merely in terms of time. The model and some psycho-physiological laws such as Fitt's law have been applied to time and motion studies during the performance of editing tasks. This tayloristic approach can be applied to optimize the interface with respect to the total amount of time necessary for task accomplishment by means of different devices, such as keys or mouse. For users this model is of little help, except for the recognition that some designers view users as brainware machines.

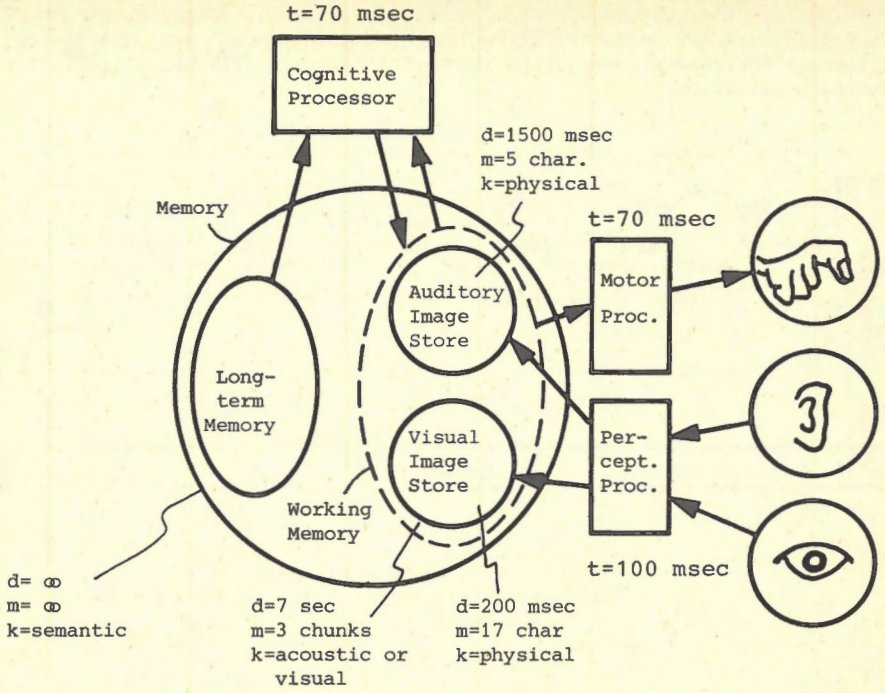


Fig. 13: Model Human Processor (slightly simplified)

The TOTE model of users (Dzida, 1982; Darlington, Dzida, & Herda, 1983) is a model of the cognitive processes inside the user. TEST, OPERATION, TEST, EXIT are the components of a feedback loop describing the control of performance in terms of cybernetics. It allows to distinguish mental activities on different levels and to provide corresponding forms of dialog. The concept of 'excursion' developed on the basis of the TOTE model describes a meta-communicative activity which can help the user to build up an adequate model of a dialog system. 'Deviations' are a means to brush up the knowledge about a model learned before or to reassure a user of the validity of his model. The resulting 'three tours model' (Fig.14) is a schema for system activities which may help the user to understand his general possibilities for acting.

Anticipating (or realizing) that users are forced to adapt themselves to the algorithmic communicating behaviour of the machine H.-J.Hoffmann has proposed to specify the future users (or at least their roles) by the same method that is also used for program specifications: by abstract data types (Hoffmann, 1983). Although this kind of interface description is advantageous for the designers it will be intransparent for most of their contractors and for naive users because of the applied modelling tool.

The 'sites, modes and trails' model (Nievergelt & Weydert, 1980) is an attempt to describe the structure of the state of the dialog system in a uniform way. The space of objects (sites), the space of activities (modes) and the history of the dialog (trails) are hierarchical structures which can be operated on by a universal set of

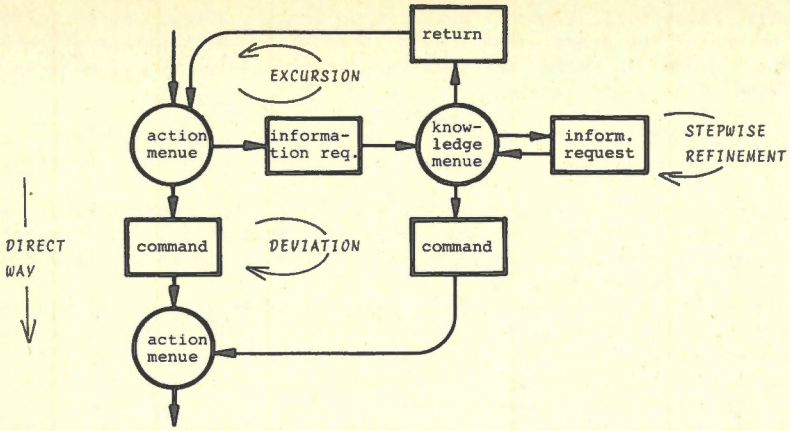


Fig. 14: The 'three tours model'

commands. Applications-specific objects and operations form subtrees of sites and modes. It is argued that the simplicity and uniformity of the whole model make it easy for a user to understand the model and to act accordingly. A part of the system's state is permanently represented on the screen in a fixed format reflecting the model (Nievergelt, 1983). This makes meta-communication concerning the model partially superfluous.

Finally, the desk-top model applied in modern personal computers (XEROX STAR, APPLE LISA) tries to bring the state of the dialog system back to the user and to let him quasi-directly operate on the visible objects. Meta-communication facilities are realized by the same techniques as for other objects and operations, by windows and pop-up menues.

All models discussed have been built by designers or researchers and are heavily used only by them. Most of the models cover only a small section and a special aspect of the whole system covered by the meta-model. Users had hardly any influence on the respective modelling processes.
Due to the traditional concentration on software production user-designer communication has not been considered an important part in life-cycle concepts for systems development. In the next section we will argue that possibilities for direct communication between designers and users as indicated in the meta-model can be a key for the improvement of HCC.

4. CO-OPERATIVE MODELLING

The discussion above has shown that one fundamental prerequisit for users' control of their working situation is still missing: a systematic procedure to build and learn adequate models of the computerized environment.
Instead of adhering to the old principle 'Know the user!' (Hansen, 1971) leaving users in a passive role, we propose to let users actively participate in system development. This can be achieved by a co-operation of users and designers using direct communication and by designing the dialog system as a transparent partner with meta-communicative capabilities. The first step must be a modification of attitudes.

Designers must leave their arrogant position of knowing always and in advance what is good or 'friendly' for users. They should learn that some of their attempts to be 'user-friendly' might be more confusing than helpful, e.g. the promise to deliver 'self-explaining' dialog systems.
The term 'self-explaining' itself is misleading:

- The explanations given are not given by the system 'itself' but are explanations provided by the designer and are based on a model he thinks to be good for the user.
- Explanations are necessarily incomplete. The whole terminology and the modelling tools applied are not explained explicitly. For example, the description of the input syntax in BNF notation is in no way self-explaining for a user not knowing BNF.
- An explanation system can hardly check whether a user has understood the explanations. Even sophisticated tutorial systems are helpless if a user makes the same error again and again. They cannot switch to a meta-level to find out the problem. The little success of computer-aided instruction should be a warning for advocates of self-explanations.

Nevertheless, a decent explanation component is helpful in every dialog system, especially after intensive training of the user.
A similar problem arises if the communicating behaviour of the machine is based on a partner model, which adapts automatically to the users' behaviour, but is unknown to the users. Since they are unable to find out by meta-communication what causes a change of behaviour, users easily adopt an anthropomorphic view of the dialog system. This tendency is increased if natural language is used for meta-communication (Kupka, 1984). A wrong partner model in the user is the result.

Users and especially their employers, on the other hand, must learn that highly sophisticated systems cannot easily be described in familar terms and used without special training - in contrast to the promises made in advertises. They should agree to learn about new modelling tools and models to be able to master a dialog system and for active participation. They should claim the right to talk to the designers directly.

To achieve co-operative modelling we propose the following procedure.

Step 1:

To be prepared for participation users (or their representatives) have to learn about some, at least semi-formal modelling tools. Nets as used in this paper are good for this purpose (Oberquelle, 1980a). They are simple graphic tools, which can be applied to different aspects of organizations in general. They allow refinements and abstractions as well as taking out substructures and embedding. Combined with other tools they can be used even for formal specifications.
That nets are a handy tool for communication between researchers, too, is illustrated by the growing number of researchers using them (cf. Dzida, 1983; Hoffmann, 1983; Tauber, 1984). Learning about nets as a tool can simply be done by using them for situations known from the user's working environment - as it was done throughout this paper.

Step 2:

Before even thinking about implementation designers and users must develop a common model of the intended work situation. Partial models for different aspects will be necessary in addition.

- One aspect is the static structure:
  The first step should be the development of a global CA-net model of the organization the dialog system will be embedded in. It will especially refine the organizational environment shown in the meta-model (Fig.8).
  Refinements of some of the subsystems must be added. They show, for example, the dialog system as an applications-independent part linked to tools and

channels for applications-dependent objects.
Extensible conventions, partner model and a subsystem for explanations as special components for meta-communication should be provided right from the beginning.

- As a basis for describing the dynamic behaviour three things are necessary:
  For each channel the type of its objects must be defined. The design may become simpler if only few types (e.g. strings and trees) are used.
  The states of each agency relevant for interaction with its environment must be found out and named carefully.
  The activities of each agency must be identified and named. Meta-communicative activities for exploration should be available in the dialog system in all its states.
- The dynamics of each agency can be defined in three parts:
  The first one describes the effects activities have on states (including switch over to other agencies). The effect activities have on objects internal to the agency can separately be described. State/transition nets and MA-nets supplemented with verbal explanations can be used for these purposes.
  Finally, the behaviour of each activity with respect to the agency's environment has to be defined. For each activity available in each state one must know the kind of information necessary for its activation and the kind if information sent to the invoker and the rest of the environment. The decision about the syntactic form of inputs and outputs can be postponed.
- The characterization of subsystems as proposed above should include a characterization of the user's role, i.e. what he is expected to do. This provides a basis for the partner model to be put into the system. The resulting partner model should be designed for user influence by letting him change its state explicitly or by asking for his confirmation for state changes proposed by the system.

Step 3:

To show the feasibility of the design and to improve the design and its models a step of rapid protoyping should be provided.
The explanation component can be simulated since all people engaged at this time know the system very well from other sources. The syntactic form of inputs and representations of objects as outputs can be tested and defined at this time.

Step 4:

The full, efficient implementation follows as the next step. Traditional methods of software engineering can be applied. Now additional provisions for modelling are necessary: The full explanation component must be carefully implemented, comprising all models of step 1 at levels consistent with the partner model inside the dialog system. It may be supplemented by examples or tutorials (cf. Tauber (1984)) and should contain a personal notebook for the user for remembering interesting details, e.g. about self-defined commands and objects. To allow immediate feedback to the designers a separate 'complaints channel' may be provided.

Step 5:

Systems installation in the organization must be accompanied by a special training for all users covering the modelling tools and the partial models developed in co-operation. Practical exercises give an introduction into the handling. Some of the designers should work as instructors to get direct feedback from the users.

Step 6:

Systems adaptation to special user needs is always necessary. Part of it can be done under direct user control by extending the conventions by new abbreviations, user defined commands and special defaults. Parallel updating of the explanations must be possible, e.g. using the user's notebook.
More fundamental modifications can only be done by the designers. Routine meetings with users and evaluation of the 'complaints channel' will help to find out problems. The models known to both parties will serve as a basis for competent discussions.

The systems development procedure proposed above contains elements vividly discussed in software engineering. They are characterized by shorter feedback loops. The need for co-operative modelling as well as for transparent dialog systems with meta-communicative functions provides new arguments for increased direct communication between designers and users.

5. OUTLOOK

The role of the researcher has not been considered so far: From our point of view the task of researchers is to support fair co-operation. First of all they can do this by interpreting 'cognitive ergonomics' in a wide sense, much wider than is reflected in the term 'software ergonomics', which is sometimes used as a synonym.
Researchers may help to develop adequate modelling tools and provide computerized systems for handling external representations of models, e.g. net editors. If the descriptions of models are stored and edited in the computer, it will be easy to put them into the explanation component of the dialog system itself.

A second field may be the development of 'model architectures' (in the sense of meaning (iv)) and 'model procedures' for algorithmic meta-communication which can be adapted to special applications, a field not understood very well up to now.
A third possibility lies in the creation of tools for rapid prototyping built on the basis of 'model architectures'.

The last, but possibly most important question is whether we have already found the proper paradigms on which modelling tools should be based.
Possibly, many difficulties have been caused by using two very different paradigms. Objects in channels inside the computer are usually treated as values ('data') which are transported by copying. Objects outside are individual things (e.g. documents) which flow from position to position.
Viewing a system as a set of agencies co-operating by the exchange of objects via channels covers both paradigms. The underlying notion of object is the crucial point.
Considering dialog systems as 'generalized editors for structured, modifiable individuals' (Oberquelle, 1980b) is one specialization of this view.

The concept of the so-called 'object-oriented languages' like SMALLTALK can be explained in this framework, too. SMALLTALK objects are individual agencies with local channels. Systems are a set of objects co-operating by the exchange of messages via a common channel. Whether these two or other notions of object will provide a better background for modelling is an interesting question.

Finally, I want to thank my colleagues Ingbert Kupka and Susanne Maaß for the many fruitful discussions we had in our research group on man-machine communication in Hamburg during the last years and which deeply influenced the ideas presented in this paper. Wolfgang Dzida and Susanne Maass provided helpful comments on a draft version of this paper.

REFERENCES

Card,S.K.,Moran,T.P.,Newell,A.(1983). The Psychology of Human-Computer Interaction. Lawrence Erlbaum, Hillsdale, London

Carroll,J.M.,Mack,R.L.(1982). Metaphor, computing systems, and active learning. IBM Research RC 9636, IBM, Yorktown Heights

Darlington,J.,Dzida,W.,Herda,S.(1983). The role of excursions in interactive systems. Int. Journal on Man-Machine Studies, 18, 2, 101-112

Dehning,W.,Essig,H.,Maaß,S.(1981). The adaptation of virtual man-computer interfaces to user requirements in dialogs. Lecture Notes in Computer Science, Vol.110. Springer, Berlin, Heidelberg, New York

Denert,E.(1977). Specification and Design of Dialog Systems With State Diagrams. in: Morlet,E.,Ribbens,D.(Eds.). Proc. International Computing Symposium. North-Holland, Amsterdam, 417-424

Dzida,W.(1982). Dialogfähige Werkzeuge und arbeitsgerechte Dialogformen. in: Schauer,H.,Tauber,M.J.(Eds.). Informatik und Psychologie. Schriftenreihe der Österreichischen Computergesellschaft, Vol.18, Oldenbourg, Wien, München, 54-86

Dzida,W.(1983). Das IFIP-Modell für Benutzerschnittstellen. Office Management, Sonderheft "Mensch-Maschine-Kommunikation, Vol.31, 6-8

Halasz,F.G.,Moran,T.P.(1982). Analogy considered harmful. in: Moran,T.P.(Ed.). Eight Short Papers on User Psychology. XEROX PARC, Palo Alto, 33-36

Hansen,W.J.(1971). User engineering principles for interactive systems. Proc. AFIPS Fall Joint Computer Conf., 523-532

Hoffmann,H.-J.(1983). Anwendung von Spezifikationstechniken auf die Komponente Bediener eines interaktiven Systems. in: Schauer,H., Tauber,M.(Eds.). Psychologie des Programmierens. Schriftenreihe der Österreichischen Computergesellschaft Vol.20, Oldenbourg, Wien, München , 211-251

Jacob,R.J.K.(1983). Using Formal Specifications in the Design of a Human-Computer Interface. CACM 26, 4, 259-264

Kupka,I.(1982). Programmstrukturen I. Lecture Notes. Universität Hamburg, Fachbereich Informatik

Kupka,I.(1984). Algorithmische Metakommunikation. in: Schauer,H., Tauber,M.(Eds.). Psychologie der Computerbenutzung. Schriftenreihe der Österreichischen Computer Gesellschaft, Vol.22 (in preparation)

Kupka,I.,Oberquelle,H.,Wilsing,N.(1975). An Experimental Language for Conversational Use; Universität Hamburg, Institut für Informatik, Bericht Nr. 18

Naur,P.(1982). Formalization in Program Development. BIT 22, 437- 453

Norman,D.A.(1981). The Trouble with UNIX. Datamation, Nov. 1981, 139-150

Nievergelt,J.(1983). Die Gestaltung der Mensch-Maschine-Schnittstelle. in: Kupka,I.(Ed.). GI - 13. Jahrestagung. Informatik-Fachberichte, Vol. 73, Springer, Berlin, Heidelberg, New York, Tokyo, 41-50

Nievergelt,J.,Weydert,J.(1980). Sites, Modes, and Trails: Telling the user of an interactive system where he is, what he can do, and how to get to places. in: Guedj,R. et al.(eds.). Methodology of Interaction. North-Holland, Amsterdam, 327-338

Oberquelle,H.(1980a). Nets as a Tool in Teaching and Terminology Work. in: Brauer,W.(Ed.). Net Theory and Applications. Lecture Notes in Computer Science, vol. 84, Springer, Berlin, Heidelberg, New York, 481-506

Oberquelle,H.(1980b). Benutzergerechtes Editieren - eine neue Sichtweise von Problemlösen mit DV-Systemen. in: Hoffmann,H.-J.(Ed.): Programmiersprachen und Programmentwicklung. Informatik-Fachberichte, Vol.25, Springer, Berlin, Heidelberg, New York, 211-220

Oberquelle,H.,Kupka,I.,Maaß,S.(1983). A View of Human-Machine Communication and Cooperation. Int. Journal on Man-Machine Studies 19, 4, 309-333

Parnas,D.L.(1969). On the Use of Transition Diagrams in the Design of a User Interface for an Interactive Computer System. Proc. ACM 24th Nat. Conference, 378-385

Tauber,M.J.(1984). Zur Spezifikation und Konstruktion von HELP-Systemen. GI Fachgruppe 'Interaktive Systeme'(Ed.). Notizen zu Interaktiven Systemen 12, 71-87

Troy,N.(1981). Zur Bedeutung der Stresskontrolle. Experimentelle Untersuchungen über Arbeit unter Zeitdruck. ETH Zürich, Dissertation

Young,R.M.(1981). The machine inside the machine: user's models of pocket calculators. Int. Journal of Man-Machine Studies, 15, 51-85

INFORMATION SYSTEMS DESIGN METHODOLOGIES AND THEIR COMPLIANCE WITH COGNITIVE ERGONOMY

Roland Traunmuller

Institut fur Informatik
Johannes Kepler Universitat Linz

1. INTRODUCTION

As methods and tools for the design of information systems have mushroomed in recent years, IFIP has carried out a review process on the methodological development (CRIS - Comparative Review on Information Systems Design Methodologies /1,2,3/). Based on the author's participation in the review process a precis of pivotal methodological approaches is given. According to the scope of this symposium /4/ topics relevant to cognitive ergonomy are stressed.

So the following sections focus on four subjects:
reasons for the development of methods and methodologies (2,3);
pivotal approaches and methods for the design (4,5);
topics of product models and process models with relevance to cognitive ergonomy (6,7);
information modelling as an example for methodological issues (8).

2. THE NEED FOR METHODS AND METHODOLOGIES IN INFORMATION SYSTEMS DESIGN

2.1. The need for methods in information systems design

In the beginning systems analysis and programming were considered rather artist-like professions. From the procedures used bey these artist-system analysis some have become techniques and subsequently in establishing theoretical background have developed into methods.

The underlying pressures leading to industrial-like production methods can be outlined as follows:
The result of the design is a blueprint of a technical product and has to fulfil certain requirements. It is a must to have the design constructed in a reliable way and by use of commonly accepted technical methods;
The design of an information system is too huge a task to be to be done by one designer alone. So it is necessary to split the global task into separate ones. Furthermore the work has to be organized and controlled, rendering each task provable and each design exchangable; No technical product of a higher degree of complexity being achieved in one single step, verifiable checkpoints marking intermediate products have to be defined;

Information systems comprise a huge investment and have high impact on organizations. It would be frivolous to have them built without first having carefully checked the blueprint. It must be evaluated against a set of criteria designating the quality of design;
Documentation being of high importance the need for a sound methodology and a common accepted terminology becomes urgent;
Computerized design aids (tools) also press for the development of methods that can be combined and concatenated via defined interfaces.

2.2 The need for a methodology covering the design methods

The fast growth of the field has resulted in a huge number of methods and also has led to terminological confusion:
many methods are ill defined and often mere techniques lacking a methodical basis;
a diffuse terminology uses the same term for different things and vice versa;
since many methods only cover small sectors of the life cycle , concatenation of methods at thoroughly defined interfaces becomes vital.

It is a law in the development of sciences that meta-methods become necessary when terminological differences and competing methods spur confusion. Some methods proposed are also called methodologies by their authors to pronounce the fact that they consist of more than one method or that they comprise very different aspects.

There are three fields of computer science from which methodologies have emerged:
System analysis: The main development has come from system analysis with techniques gaining a more concise and powerful background;
Database design: Information systems are marked by their abundance of data and database design has attracted major attention. So many methods start with the description of objects and a definition of the data representing them;
Software engineering: There is an overlap between information systems design methodologies and software engineering with both fields comprising requirements definition and specification.

3. BASIC CONCEPTS USED IN INFORMATION SYSTEMS DESIGN METHODS

Very early basic concepts have been established (cf. /5,6/ for an elaborate description):
life cycle, a concept defining phases with exact interfaces breaks down the overall task in better manageable pieces;
abstraction, a process intended to reduce complexity by stepwise development excluding less relevant features;
database levels, a concept developed by ANSI/SPARC, is intended to reduce the complexity of the data world;
project management, a procedure ensuring planning and decision and enforcing control in the design process.
These concepts build a rough guideline for the design process providing an adequate framework for description as is shown in figure 1. Recent investigations

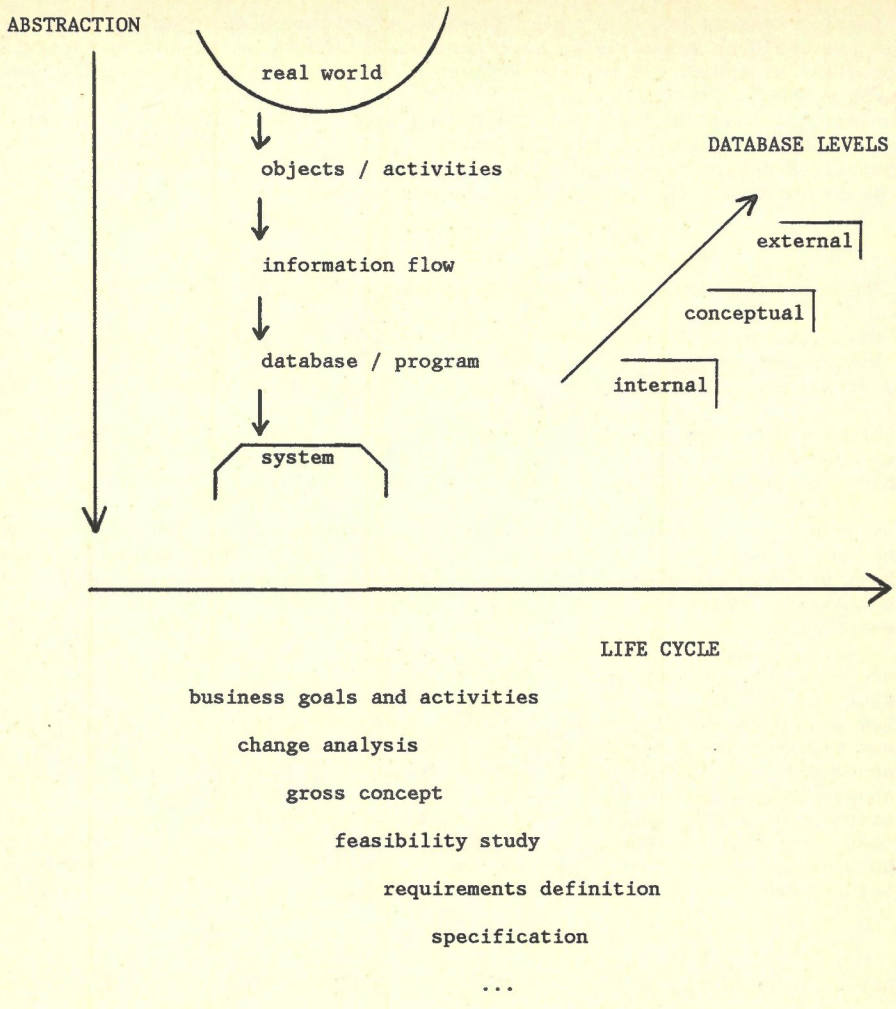


Figure 1 : Basic concepts used in design methods
(cf. section 3)

give new insight into these concepts and the design process is attracting new interest as will be discussed in section 6.

4. PIVOTAL APPROACHES OF INFORMATION SYSTEMS DESIGN METHODS

4.1 General remarks

A wide variety of methods has been covered by the CRIS review /1,2,3/ and further surveys (eg. /7,8,9/). It is difficult to classify because some boundaries are artificial and many methods converge by a vivid cross fertilization of ideas. Looking at the basic assumptions five mainstreams may be recognized:
procedure oriented approach;
data oriented approach;
behaviour oriented approach;
prototyping approach;
logical modelling.
The basic assumptions underlying this categorization will be discussed in the subsequent sections using figure 2.

4.2 Procedure oriented approach

Methods routed in systems analysis techniques usually start with the description of the business functions that are then broken down hierarchically. Functions described as boxes on higher abstraction level are detailed on lower ones. By adopting the requirements points of view and extensively using graphical representations, procedure oriented methods have proved themselves as excellent vehicles in communication to the non-professionals. But lacking formalization they are less prone to automatization. There are well known methods using the procedure oriented approach:
ISAC /10/ that will be sketched as an example in section 5;
SADT /11/ that has gained very broad practical experience.

4.3 Data oriented approach

In the data oriented approach main interest is put on static properties and the conception of database. From there subsequently business functions are modelled in interpreting the functions as transactions changing the database. So the data oriented approach is somewhat complementary to the procedure oriented one where the designer starts with the business functions and at last comes up with a data model.
Due to the fact of more than a decade of database research many techniques have been developed, some of them turning into elaborate methods:
NIAM /12/ uses a binary data model and will be shown as an example; ACM/PCM /13/ is a very elaborate database design method on the verge of the behavioural approach.

4.4 Behaviour oriented approach

In the behaviour oriented approach the basic concepts are events triggering state transitions on certain conditions. There are some similarities between

this approach and the data oriented one. The main difference is the interest in dynamic properties- while the data oriented approach is only interested in states and transitions relevant to the interpretation of business functions, the behaviour oriented approach tries to comprise the totality of possible transitions and comes close to such concepts as data encapsulation and abstract data types. As a result of the sophisticated proceeding the specification becomes very voluminous.

PSL/PSA /14/ may be judget as an early forerunner of this approach. It has gained some acceptance in practice but also lost theoretical interest on grounds of conceptual inconsistencies. Further methods are:
REMORA /15/ that will be used as an example;
IML /16/ a Petri net formalism on state conditions;
ACM/PCM /13/ already mentioned in the preceding section.

4.5 Prototyping as an experimental approach

Prototyping is the procedure of constructing a crude system in an experimental way. There are two goals: having a crude version, neat specifications may be drawn; and by showing the system to the user requirements can be inspected. Both advantages are sometimes debated: Will the user abandon a running prototype? Can correct specifications be derived from sloppy ones? Prototyping includes some more and less methodical attempts:
USE /17/ builds an interactive information system based on a relational database and stresses the first point;
BOP /18/ uses APL-modules for rapid prototyping and is dedicated to the second goal.

4.6 Logical modelling at the conceptual level

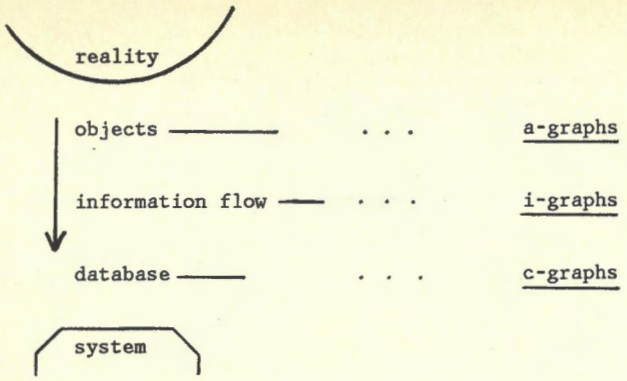
It is evident that some advantages of prototyping might also be achieved by inspecting in a model the logical structure of the system in question. So conceptual modelling during early phases of a project has been suggested as a mode of mental prototyping. The paper-and-pencil-prototyping has strong and weak points as well: an advantage is surely avoiding having to build the actual prototype; having no actual prototype misses the opportunity of having the user interface checked.

Logical modelling on the conceptual level has been suggested by Bubenko in CIM /19/. His proposals are important in an additional way by establishing links between design methodologies and the wide fan of developments in AI (eg. logical programming with PROLOG).

5. A SKETCH OF SOME PIVOTAL DESIGN METHODS

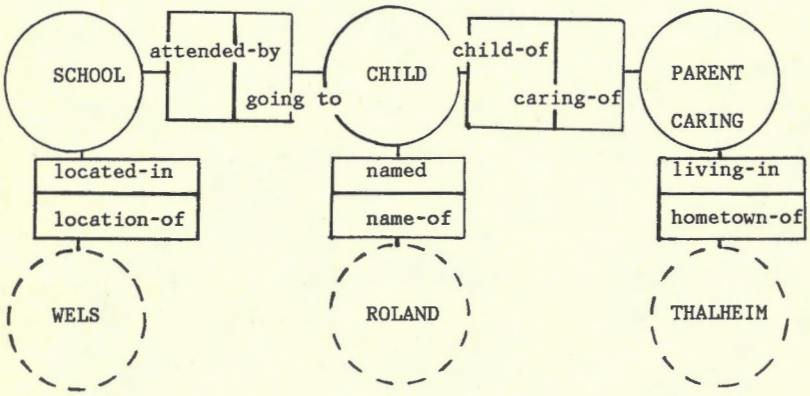
5.1 General remarks

In the following some pivotal design methods corresponding to the approaches discussed in the previous section are described. Due to the fact that in the literature /1/ each method needs a voliminous description of fifty pages or

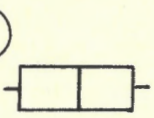


Infological model — Constructs in ISAC

a) Procedure oriented approach

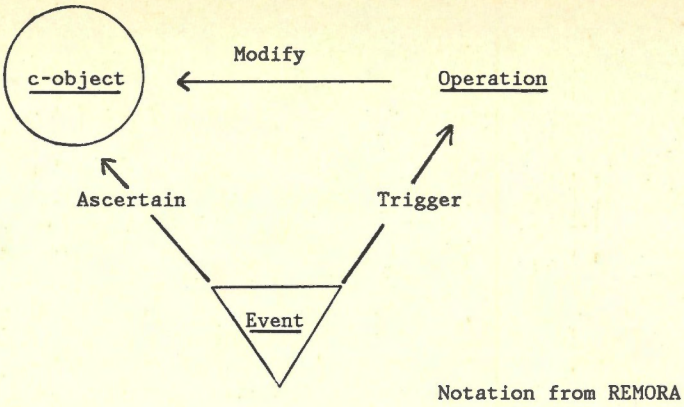


Notations in NIAM: LOT (Bridge- and Idea-type) NOLOT

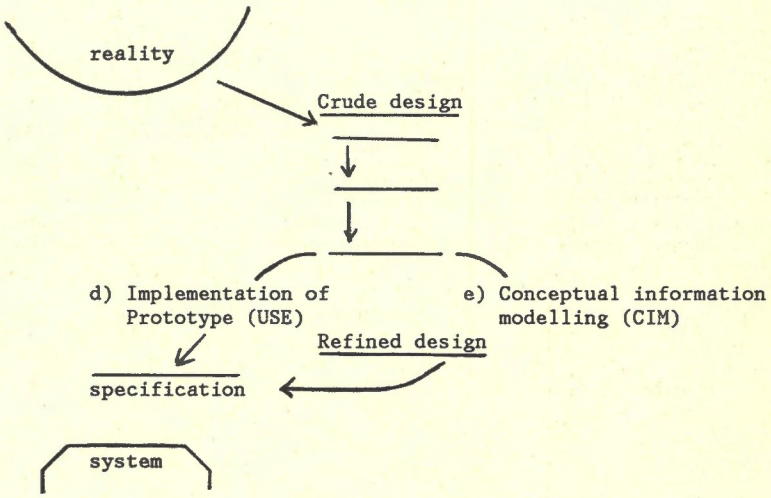


b) Data oriented approach

Figure 2 : Pivotal approaches and methods
(cf. section 4 and 5)



c) Behaviour oriented approach



d) and e) Prototyping and Logical Modelling approach

Figure 2: continued

more, the following lines can only give a very rough impression of some basic features (cf. figure 2).

5.1 A procedure oriented approach: ISAC

ISAK /10/ has been developed by the Langefors school based on the original infological approach. According to it three graphs are constructed representing the levels:
A-graph for activities at the object system level;
I-graph for the information flow;
C-graph for components at the datastructure level.
The ISAC method deals very thoroughly with the activities on the object system level to enforce change analysis. At the beginning current problems, situations, and needs are studied to prepare change alternatives. The authors stress the fact that change analysis in some big projects led to abstaining from further computerization.

5.3 A data oriented approach: NIAM

NIAM /12/ has been developed by Nijssen as a comercial product for CDC. It is based on a data model with binary associations that has also been proposed by ISO. The basic procedures in NIAM are the following:
describe the object system;
describe the information flow in elementary sentences;
transform these elementary sentences in a data model with relation- like binary associations;
represent the data model to the non-professionals by graphs and to the system builders by a specification language (RIDL).
The example in figure 2 is drawn from the regulations on school commuting allowances. According to Austrian law a subsidy is granted to those parents caring for children attending a school outside home location /20/.

The transformation from sentence to association is crucial and so NIAM has a sentence model enriched by expressions for types and constraints:
LOT: Lexical object type like surname, town-name, etc.;
NOLOT: Nonlexical object type like person, town, etc.;
Roles: Predicates like "lives-in";
Constraints: Identifiers, subsets, equality, uniqueness, disjoint.

5.4 A behaviour oriented approach: REMORA

REMORA /15/ has been developed by Colette Rolland at the Sorbonne with the industrial background of Thomson-CSF. In REMORA the following distinctions are essential:
Categories of phenomena: objects, events, operations;
Categories of associations: Modify, ascertain, trigger;
Categories of relation types: C-object, c-operation, c-event;
Static subschema: C-objects;
Dynamic subschema: C-operations and c-events.

5.5 A prototyping method: USE

USE /17/ has been developed by Wassermann at the university of San Francisco and is intended to introduce prototyping as an aid to analysis and specification of interactive information systems. The prototyping is supported by some computerized tools:
TDI: Transition diagram interpreter modeling the user dialog;
TROLL: Relational database management system;
USE Control System: A support for the coupling and design of various modules.

5.6 A logical modelling method: CIM

CIM /19/ delivers a high level conceptual description in a language with a predicate-calculus-like syntax. Objects are described using types and an elaborate time model.

6. DESIGN PRODUCT AND COGNITIVE ERGONOMY

6.1 Product models

The goal of the design is a product model that can be used as a blueprint for the building of the system. Hence the product model is aimed at three different goups of persons:
The builders who use the product model as a blueprint for implementing the final system;
the managers of the enterprise who formulate the objectives and provide the money;
the future end-users who hopefully have their representatives included in the discussion.

6.2 Constituents of a good product model

Design methodologies have put main emphasis on the product model. A wide spread of desirable features has been formulated in /2/. They can be summarized in four clusters of features.

Technical soundness: Technical soundness is a sine qua non of the design process and comprises various competing attributes. Up to now technical soundness has been a prior concern in design methods and has been reflected in such questions as:
Check whether the specifications meet the requirements stated?
Are there design flaws and incompleteness in design?
How robust is the design?

Economic success: Economic reasons have an overall stimulus on the development of methods, but it is difficult to allocate gains. Nevertheless methods feel an economic drive:
the broader the use of tools the more urgent the methodical background; overall gains often can only achieved by a concatenation of methods that is inversely dependent on progrees in methodologies.

Organizational synergy: The succes of early system analysts was largely based on their intuitive feeling in juxtaposing technology and organization. This is no longer the case in computer aided methods, where the intuition has to be replaced by new ways of socio-technical engineering.
Large information systems might overwhelm any solution based on intuition; new technical systems, as eg. office systems, are too complex to be constructed in a way that is adding technology to organization.

Cognitive ergonomy: Although questions of organizational ergonomy very early struck the developer's mind, cognitive ergonomy only recently has gained attention. The subsequent section will deal with some issues that link information systems design methodologies with cognitive ergonomy.

6.3 Topics relating design products to cognitive ergonomy

Regarding design as a product four questions touching cognitive ergonomy can be located:
communication with the builders;
communication with the managers;
communication with the endusers;
visualisation of the information.
These issues are covered in a different way by the diverse approaches. In figure 3 an indication is given showing the level of consideration attained.

Communication to the builders: Design as a blueprint is primarily targeted at the builders. Questions of communication have been tackled since the very beginning. Diagrams, graphs, and prose are widely used with more formalized methods gaing ground. It is the old dream of automatic programming that by feeding a specification into an automaton the builders/implementors become obsolete.

Communication with the managers: System analysts alway have been aware that communication with the clients has the highest priority. A lot of graphic methods and techniques have been developed. But there is a serious pitfall in all graphical methods, sometimes nicknamed "boxology". Everything can be written into a box or close to an arrow - no consistency is checked or guaranteed. As a consequence there is a trade-off in every method between the communication with the builder and the communication with the clients.

Communication with the end user: A different type of client is the end user. It is essential that the gross product model of the future information system is also presented to the prospective end users. User participation is intended to ensure a representation of end users in an institutionalized way. There is an intrinsic problem: the end users have to be envolved in an early stage, but they need elaborate models of the future system to anticipate its acceptance. Piloting and prototyping have emerged as an answer to this problem.

Visualisation of the information: Planning the detailed visualisation as cited in the last paragraph is beyond the scope of most methods. The only exception is prototyping.

	ISAC	NIAM	REMORA	USE	CIM
communication with the builders	*	**	**	*	**
communication with the managers	**	*	*	**	*
communication with the end users	*			*	
visualisation of information				**	
evolutionary design	*			**	**
trial and error				*	**
abstraction	*	*	*		*
assistance by tools	*	*	*	**	
linkages to AI		*			**

Figure 3 : Topics relating information systems design methodologies to cognitive ergonomy (cf. section 6.3 and 7.3)

The marks blank, asterisk, double asterisk indicate the level of consideration gained.

7. DESIGN METHODS AND COGNITIVE ERGONOMY

7.1 Process models

The term design is ambiguous, meaning both the product and the process. From the beginning main interest has been directed towards design as a product reducing the aspects of the process to a basic framework of life-cycle, abstraction, database levels, and project management (cf. section 3). Now growing attention is paid to models of the process(eg. /3,21/). Thus new questions are conveyed into the researcher's scope, reflected by the establishment of an IFIP task group dedicated to the topic of understanding the design process /22/.

7.2 Prescriptive and descriptive process models

Prescriptive process models lay down how the designer shall proceed. The basic framework has been extended or detailed in almost every method. For the abstraction process some extensions already have been mentioned:
ISAC: an elaborate activity level focussing on alternatives;
NIAM: an information flow mapped in a sentence model;
CIM: a conceptual specification preceding the classical design.

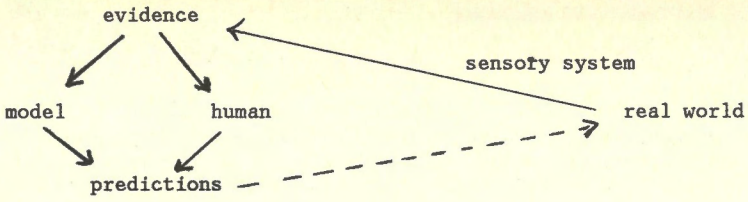
Descriptive process models describe what the designer actually does when he designs. Descriptions are based on the level of behaviour with some attempts to investigate the underlying cognitive processes. First findings indicate a severe deviation from the way hitherto claimed as normative. Nearly every project has its individuality forming its own ways of organizing, proceedings, documentations, and standards. The result makes obvious a categorization of process models:
scale of the information system; type of the problem as eg. technical or administrative;
type of the system as batch, online, distributes, etc.;
start from scratch or embedded system; skill of the designers.
Investigation in de facto design has put strong arguments in favour of the cognitive model chosen first being decisive for the subsequent process and giving only "token service" to alternatives /23/. This result finds its correspondence in cognitive models of problem solving /24/.

7.3 Topics relating process models to cognitive ergonomy

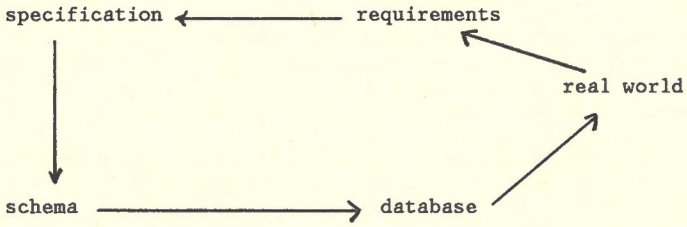
Figure 3 also includes those topics that relate design as a process to cognitive ergonomy.

Evolutionary design: It is important to plan the process evolutionarily. Design is an open-ended activity aimed at the building of adaptable systems. Evolutionary design takes into consideration the fact that the users are unable to express all their needs and that the requirements are constantly shifting as well.

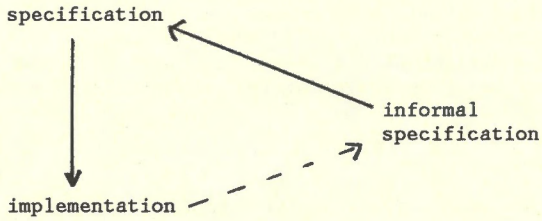
Trial and error: Even if all requirements stated before are observed, the final product may not find acceptance. Trial and error, a basic strategy for solving small problems.



a) Artificial intelligence



b) Database design



c) Programming languages

Figure 4 : Different view of the abstraction process (cf.section 8.1)

Abstraction: Information modelling at the conceptual level is the core part of the abstraction process. Some questions recently gaining attention are discussed in the subsequent section 8.

Assistance by tools: The transparency of the design process can be largely enhenced by adequate visualisation and documentation. For both points computer assistance is the only way of doing it.

Linkage to artificial intelligence: Taking into account the spectacular developments in artificial intelligence, methods that can be linked are going to gain an advantage. The advantage is mutual: design methods can be improved and artificial intelligence may use design methods in tackling large systems.

8. INFORMATION MODELLING AS AN EXAMPLE OF METHODOLOGICAL ISSUES

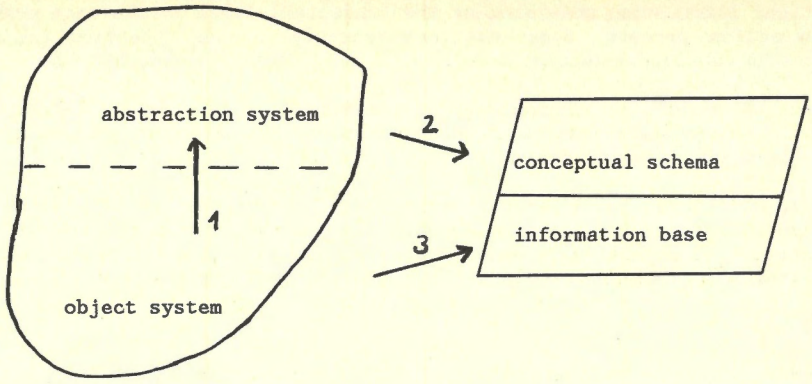
Information modelling is the core part of the design process and has gained vivid consideration in all methods cited. There are some meta issues arising that will be discussed in this last section: the different view of the abstraction process in artificial intelligence, database design, and programming;
the proposal of a reference schema by ISO;
the abstraction system as the mental picture of the universe of discourse.

8.1 Different views of the abstraction process

The process of information modelling has been seen differently in artificial intelligence, database design, and programming /25/ (cf. figure 4):
Artificial intelligence is concerned with modelling human thinking. The model and the human both make predictions about the real world. The model is successful if the predictions are indistinguishable from the human's prediction (Turing);
Database modelling emphazises states with peoples as agents of transactions. Databases comprise large amount of formatted data, the meaning of which is well understood;
Modelling in programming is concerned with the engineering of software and sees programs as agents implicitly defining states.

8.2 A reference schema for information modelling proposed by ISO

Every method in question has its wayof abstracting. So the International Organization for Standardization has put forward a proposal for a reference schema /26/ (cf. figure 5), comprising:
the universe of discourse, as the portion of the real world to be modelled;
an abstraction system, including the classes, rules, etc. of the UoD;
an object system, as the part of the UoD not contained in the abstraction system;
a conceptual schema, as the description of the abstraction system;
an information base, as the description of the object system;
a universe of discourse description, comprising the conceptual schema and the information base.



universe of discourse
description

universe of discourse
(UoD)

Processes:
(1) Classification, abstraction, generalization, rules
(2) Representation of the abstraction system (writing down the rules)
(3) Representation of the object system (recording facts)

Figure 5 : A reference schema for conceptual modelling proposed by ISO (cf. section 8.2)

8.3 The abstraction system as the mental picture of the universe of discourse

The abstraction system is extracted from the object system in building classifications, abstractions, generalizations, rules, etc. This is a human process establishing a shared mental picture of the universe of discourse. It is an important question whether this should be done by the professionals or the specific field of application. The author's investigations in the field of modelling legal regulations /27/ have led to the conclusion that up to now the engagement of the computer scientist has been the decisive factor in every successful model. In the future there must be a change because it should be the layman who sets structures and definitions in his own field. Hopefully, education in computer literacy and the institutionalization of user participation will shift the balance to the middle.

REFERENCES

IFIP CRIS 1: Conference on Comparative Review of Information System Design Methodologies, Noordwijkerhout, May 1982, Proceeding T.W. OLLE, H.G. SOL, A.A. VERRIJN-STUART (Eds.), North-Holland, Amsterdam, New York

IFIP CRIS 2: Conference on Feature Analysis of Information System Design Methodologies, York, July 1983, Proceedings T.W. OLLE, H.G. SOL, C.J. TULLY (Eds.), North-Holland, Amsterdam, New York

IFIP CRIS 3: Intermediary Report on the IFIP CRIS 3 Task Group, (Chairman: T.W. OLLE), Linz, September 1984, Preliminary report

MIND AND COMPUTERS: Second European Conference on Cognitive Ergonomics, September 1984, Gmunden/Austria, Proceedings T.R.G. GREEN, M.J. TAUBER, G.C. VAN DER VEER (Eds.), Informatik-Fachberichte, Springer, Berlin, Heidelberg, New York, Tokyo

H.J.SCHNEIDER: Lexikon der Informatik und Datenverarbeitung, Oldenburg, Muenchen, Wien, 1983

P.LOCKEMANN, A. SCHREINER, H. TRAUBOTH, H. KLOPPROGGE: Systemanalyse, Springer, Heidelberg, New York, 1983

M.PORCELLA, P. FREEMAN and A.I. WASERMANN: Ada Methodology Questionaire Summary, Software Engineering Notes, 8, January 1983

H.BALZERT: Methoden, Sprachen und Werkzeuge zur Definition, Dokumentation und Analyse von Anforderungen an Software- Produkte, Informatik-Spektrum, 4, August and October 1981

P.T.M.LAAGLAND: Modeling in informations systems development, Akademisch Proefschrift, Vrije Universiteit te Amsterdam, 1983

M.LUNDEBERG: The ISAC Approach to Specification of Information Systems and its Application to the Organisation of an IFIP Working Conference, in /1/

D.T.ROSS and K.E. SCHOMAN: Structured Analysis of Requirements Definitions, IEEE Trans on SE, January 1977

G.M.A.VERHEIJEN and J. VAN BEKKUM: NIAM: an Information Analysis Method, in /1/

M.BRODIE and E: SILVA: Active and Passive Component Modelling: ACM/PCM, in /1/

D.TEICHROEW and E.A. HERSHEY: PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems, IEEE on SE, January 1977

C.ROLLAND and C. RICHARD: The Remora Methodology for Information System Design and Management, in /1/

G.RICHTER and R. DURCHHOLZ: IML-inscribed High-Level Petri Nets, in /1/

A.I.WASSERMANN: The User Software Engineering Methodology: An Overview, in /1/

O.JORDANGER: BOP Prototyping-User's Guide, SINTEF-Report STF17/A811012, SINTEF, Norway, 1981

M.R.GUSTAFSSON, T. KARLSSON and J.A. BUBENKO, JR.: A Declarative Approach to Conceptual Information Modelling in /1/

R.TRAUNMULLER: Lecture Notes in Administrative Data Processing, Universitat Linz, 1983

G.RZEFSKI: Some Philosophical Aspects of System Design in R.TRAPPL (Ed.): Cybernetics and Systems Research, North-Holland, Amsterdam, New York, 1982

IFIP TC 8: Report on the Task Group "Understanding the design process" (Chairman: R. TRAUNMULLER), London, April 1984, Preliminary report

G.RZEFSKI: private communication

M.J.TAUBER: Programmieren und Problemlosen - eine Analyse der begrifflichen Bedeutung in der Psychologie wie in der Informatik, in H. SCHAUER, M.J. TAUBER: Psychologie des Programmierens, Oldenburg, Muchen, Wien, 1983

ACM: Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling, Pingree Park, Colorado, June 1980, Sigmod Record 11, February 1981

ISO: Concepts and Terminology for the Conceptual Schema, Preliminary Report ISO TC97/SC5/WG3, ANSI, New York, 1981

R.TRAUNMULLER: Methoden zur Entwicklung von Anwendungssystemen: Eine Antwort der Informatik auf die Anforderungen neuer Anwendungen, in: R.TRAUNMULLER, H. FIEDLER, K. GRIMMER, H. REINERMANN (Hrsg.), Neue Informationstechnologien und Verwaltung, Informatik- Fachberichte 80, Springer, Berlin, Heidelberg, New York, Tokyo, 1984

INTRODUCING STATISTICAL COMPUTING - EVOLUTION OF THE COGNITIVE SYSTEM OF THE NOVICE USER

Gerrit van der Veer*, Bert van Muylwijk+, Jan van de Wolde+

*Vrije Universiteit, Amsterdam
+T.H. Twente, Enschede
Netherlands

The project that is reported in this paper is a pilot study. The main goal is the development of a method to investigate real life situations in which novices receive an introduction to a computer system, determining the interrelations between:
- characteristics of the student;
- strategy and content of the course;
- development of a model of the system in the student's mind.

If this investigation takes place in an existing course the investigator is normally not allowed to interfere with the teaching method or with the curriculum. The investigation therefore will not have the character of an experiment, the testing of hypotheses will not be a central goal. A description the phenomena is the main result, the teaching strategy (and the subject matter of the course, the task domain for which the system is introduced) not being a variable in the individual study. The results of these sources may only be speculated upon, albeit that comparison with similar studies in the long run will result in clarity about the relations (Eason, 1983). The authors plan to repeat this study with a number of different institutions for other task domains.

1. INTRODUCTION

1.1. Models

The proliferation of digital computers in the past few decades coincided with a paradigm shift in psychology from a behavioural to a cognitive approach. At the very heart of the latter approach lies the conviction that human behaviour cannot be properly understood or predicted without paying attention to the way the outside world is being represented inside the head of the agent. Just like the output of a computer, the behaviour of a human agent cannot be explained solely on the basis of the input. You need to know how incoming data are being interpreted and processed in order to account for the variation in responses to these data. Cognitive psychologists have introduced numerous concepts to refer to such processes, e.g. schemata, semantic networks, mental models, entailment structures. This variety in terminology gives way to a lot of confusion since many of these (hypothetical) constructs share connotations. To add to this confusion labels like 'mental models' are used by different authors to refer to different matters. In discussing the role of mental models in the process of man-machine interaction Norman (1983) offers a nomenclature that we will adopt here just for the sake of clarity. He distinguishes the target system, the conceptual model of this target system, the users mental model of the target system, and the cognitive scientist's conceptualisation of that mental model. Conceptual models are invented and used by teachers, designers and engineers as opposed to mental models, that evolve naturally through interaction with the target system. It is not really clear whether mental models can be conceived of without making implicit references to the scientist's conceptualisation of

these. But we will not go into that question any further. Observations on mental models bring Norman to the conclusion that "most people's understanding of the devices they interact with is surprisingly meager, imprecisely specified, and full of inconsistencies, gaps, and ideosyncratic quirks". People in the field of cognitive ergonomics will arrive at similar conclusions either looking back at their own computing history or observing their subjects.

1.2. Individual differences

There is much variation in the nature and the adequacy of the mental models that guide the behaviour or computer users. Among the possible sources of variation the level of experience has obtained most attention from researchers. Much of this research draws upon well known studies regarding differences between novices and experts in solving chess problems or science problems (e.g. McKeithen, 1981). The general conclusion is that experts do not only know more of computing, but that their knowledge is also better organised and (by consequence) more easily accessible. Other authors (e.g. Kahney, 1983) established however that the level of experience is certainly not the only source of variation in the organisation and amount of computing knowledge. Individual differences have shown to be due to several other factors such as personality traits and dimensions of cognitive style. In one of our earlier experiments we found that the acquisition of computing concepts, such as conditional branching, is significantly effected by learning styles as indicated by Pask's serialism-holism dimension (van der Veer and van de Wolde, 1983). We intent to explore the role of several other dimensions of personality and cognitive style that look promising for the prediction of phenomena of human computer interaction (van Muylwijk, van der Veer and Waern, 1983). One of these variables has been included in the pilot study reported here (impulsivity-reflexivity).

1.3. Metaphors

An important means of establishing adequate mental models of computers and computing in novices is the use of metaphors. Metaphors are very powerful in activating existing schemata that may help to grasp critical features of computing systems. They may offer a shortcut to prevent teachers from digressing on computer architecture. Peelle (1983) describes different categories of metaphors and related approaches to be used in education about computers. Lawson (1982) presents elaborate examples of some metaphors, intended for a first introduction to computer systems, which have been the basis for the analogies used in our study.

2. A PILOT STUDY

The first opportunity to try out our approach was in of an existing introductory course on statistical computing at the Department of Social Sciences at the Vrije Universiteit in Amsterdam. The format and the content of the existing course had to be kept intact, but the teacher was very cooperative in providing opportunities to make observations and to administer special measurement devices, and in discussing his teaching strategy with the investigators before the course took place. In fact this is a condition to be able to construct the relevant questionnaires about the changing models of the system we are interested in, and which are dependent upon the actual structure of the course.

As stated in 1.2. an important source of variation is the domain of individual differences between novice students. There are two aspects that we would like to consider. The first one deals with the experience of the student with computation and with computer systems, and his beliefs and attitudes in this field. The way the values of these variables are established will be treated in 2.1.2. The second aspect concerns the more stable modes of cognitive functioning that are expected to

be relevant to the population of students concerned: the domain of cognitive styles. In the present study, however, we were obliged to restrict ourselves to one variable in this field, that could be measured in a relatively short amount of time, and in fact we used one that was intended to provide the students with an actual database to practise with during the course. This domain is therefore only partly covered. The results were not expected to be very impressive, but the inclusion of at least one cognitive style dimension would suffice to illustrate the method we propose.

2.1. Method

2.1.1. Subjects

24 University students, mainly from the Department of Social Sciences, enrolled for a course introducing computational statistics. None of them had any knowledge of the computer system to be taught, all had a background in applied statistics. All had been at the university for at least four years. The students were told the aim of the study at the start of the project, and had the freedom to refrain from participation. Although no one refrained from participation in the study as a whole, a few students did not take part in some of the tests during the course. All students chose their own identification number from a list, and were anonymous to the investigators, to the teacher and to their fellow students.

2.1.2. Measurements prior to the course

(a) computer literacy
The experience, attitudes and beliefs of our students in the domain of computer use are measured with a translated and abridged version of the Minnesota Computer Literacy and Awareness Assessment (Anderson et al., 1979). The questionnaire takes a maximum of 25 minutes. The resulting scores are:
- CASTkp, procedural knowledge;
- CASTkc, conceptual knowledge;
- CASTam, attitude towards mathematics;
- CASTas, attitude towards science;
- CASTx, amount of experience with computers;
- CASTed, amount of education about computers.

The scoring we used is an adaptation by the authors. Only CASTkp and CASTkc are part of the original scoring method. The other indexes resemble scores that Anderson used in experimental studies only. We used an abridged version of the test, that in one of our other studies showed a correlation of .91 with the complete version, for a sample of subjects comparable the subjects reported on here.

(b) cognitive style
As mentioned in 2., there was no opportunity to invest a lot of time in measuring cognitive styles. The test we included as an example is the IRT, an experimental version of an impulsivity-reflexivity speed test, developed at the Vrije Universiteit of Amsterdam. The test resembles the MFFT by Kagan (Kagan et al., 1964). The items are designed to enable a scoring method aimed at measuring the speed-accuracy trade off, unrelated to cognitive capacity. The test takes a maximum of 20 minutes. The test results in two relevant scores:
- IRTe, number of errors;
- IRTt, total solution time.

In our sample the correlation between these two indexes was negative, as predicted (-.54). The first index however turned out to be very unreliable, whereas the second one had a split half reliability coefficient of .93 (corrected for test length: .96). For our analysis we will therefore only use IRTt.

2.1.3. Physical arrangement of the course

The course took place in a special classroom where the students were seated in pairs behind 12 terminals connected to the central university computer system, a configuration with two Control Data Cyber 170-750 mainframe systems. The course lasted two weeks, about 6 hours a day. Class introductory lectures and explanation of how the computer worked were alternated with short periods of individual experimentation and practise. During 2 hours a day periods of free exploration and hands on experience were possible, on which occasion two assistants were present for additional help. In the central lectures the teacher made use of a blackboard, an overhead projector and a terminal connected to the computer system, with the display image projected on a large screen.

2.1.4. Contents of the course, design of the field study

Since the principal method of our study is the observation of phenomena and the measurement of change within the models during the course, a description of the method cannot be given without dealing with the details of the course.

(a) The first week was devoted to learning how the system works (the batch system NOS-be en the interactive system INTERCOM), including facilities like editing and text processing, sorting and file manipulation. The didactic method was centered around a series of metaphors that were deviced in order to facilitate the development of valid models of aspects of the system:
- conversation with the system was compared to conversing with a slave in a restricted language, referring to a knowledge base upon which the partners agree;
- operating the computer was compared to operating a dish washer in a restaurant;
- queue and stack operations were illustrated with the metaphors queueing for a bus and shunting operations;
- file manipulations were treated analogous to the functions of a video recorder;
- for the handling of masterfiles a library was the metaphor;
- sorting procedures were first introduced by presenting a problem about a slave who is to put a list of numbered units in their correct order, and who is too stupid to find a handy way of doing so;
- for the editor and text processor the slave was again the analogy, being a faithful servant who needed short and exact commands.

For some of these metaphors (the dish washer, video recorder - file manipulation, the sorting slave, the editing slave) we collected systematic observations: Before the analogy was introduced, we administered a questionnaire about the model of the "naive" user (see figure 1 to 4).

Design a kitchen machine for a restaurant, with the functions:
a. washing up,
b. drying.
Both processes will have a restricted capacity, e.g. 60 dishes (a) and 20 dishes (b), and last a certain amount of time, e.g. 10 and 5 minutes. The restaurant owner possesses a limited number of dishes, so he will have to wash up at regular times.

Which options for choice have to be built in:
before process a .....
between a and b .....
after process b .....

Which decisions are better left to the staff ?
.....

Which questions will the machine have to ask the operator regarding requirements (e.g. detergents) ?
.....

Figure 1 Question asked before introduction of the concept "automaton".

If you were designing a video-apparatus, which buttons would you include - think of the possibilities needed to spool to another scene, to copy parts of a tape for assembling etc.

knob	function description
.....	.....
.....	.....
.....	.....
.....	.....

Figure 2 Question asked before introduction of file manipulation procedures.

Write instructions for a slave to arrange the following group of numbers in ascending order:

3,12,-2,11,10,10,-18,0,1,0,25

Formulate the commands in such a way that they may be applied to other groups of numbers.

Figure 3 Question asked before introduction of sorting.

Write instructions for a slave to correct the indicated errors in the following poem. Use short commands but take care to be very precise.

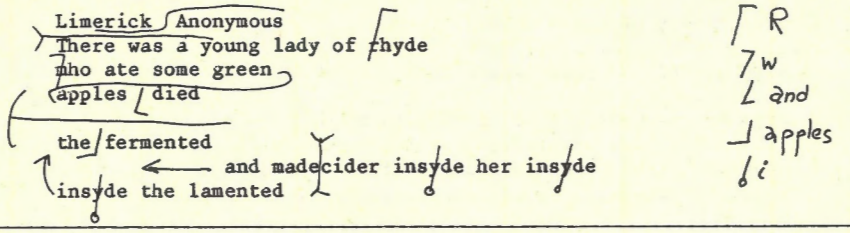


Figure 4 Question asked before introduction of the editor.

After the completion of each questionnaire the forms were collected and the instructor posed the same question in the group and collected the responses on the blackboard. With this list he started his elaboration of the model, pointing out the correspondence with computer system operations and the dissimilarities. The diversity of reactions resulted in a rich metaphor, with of course a number of useless associations that had to be discarded. This metaphor was frequently used during the rest of the course to assist in the explanation of a number of system functions, and to answer questions that arose during the next practise period. Mainly the semantic aspects of the actual system commands and facilities were treated. The syntax was only briefly mentioned, documentation was distributed and the students were advised to gain experience using the system itself.
At the end of the week a post test was administered to measure the newly developed model of the system on the same topics. For the sorting process and the editing and text processing systems the questions were nearly identical to the ones asked before the treatment. The other questions are displayed in figure 5 and 6.

Explain to a fellow student who missed the lesson concerned how to place a job that is prepared in the editor, in the input queue,
.....
explain the next thing that will happen to this job,
.....
explain when and how one knows that the output can be collect,
.....
explain how this may be arranged.
.....

Figure 5 Question about the computer system, asked after the first week.

Suppose you have at your disposal an arithmetician who knows about statistical calculations. He does not know how your data are structured. Write down a short and precise description of the way in which he can find the right numbers in your table in order to make the calculations.

Scores have been collected from 300 cases concerning the variables age (A), length (L), weight (W) and sex (S), recorded in a table with column headings A, L, W and S and rows 1 to 300. Scores are noted in years, centimetres, kilograms and 1 (women)/ 2 (man). If a score is missing a dash (-) is written.

	A	L	W	S
1	43	164	-	2
2	26	172	84	2
3	-	168	58	1
.				
.				
.				
300	56	173	-	2

Instruct your arithmetician to calculate the product-moment correlation coefficient between length and age.

.....

Instruct him to draw a frequency distribution of weights.

.....

Have the scores for weight changed into codes 1 (weight less than 60 kilogram), 2 (weight between 60 and 70 kg.) and 3 (weight over 70 kg.).

.....

Figure 7 Fragment of questionnaire about statistical computing procedures, administered prior to SPSS course.

Since all students had finished courses in theoretical statistics and courses in methodological topics, we might expect them to be able to provide these instructions for their slave arithmetician. We computed a score (STATpr) of correctness, consisting of the accumulation of indexes which indicated a mentioning of the procedures to be executed, the variables to be used, the number of cases to be processed, the identification and handling of missing data.
The SPSS course started with an introduction about a research project (actually about the IRT administered at the beginning of the first week). A number of methodological and applied statistical questions were raised in a group discussion. The teacher made sure that some of the questions that "spontaneously" arose could only be stated precisely after other problems were solved. The list of questions on which the group agreed was used as a guideline for determining the type of exercises needed to practice working with those SPSS elements covered in the course. Each element was taught in the group with using a blackboard, and afterwards reviewed via fragments of a video course deviced by Cooper (1982). Here again the syntax was only briefly mentioned. Students had to work through the manual and gain experience using the system. At the end of the second week we computed a score for the procedural model of the SPSS package (STATaf). The measurement device consisted of the same items as STATpr.

2.2. Results

2.2.1. Observations

During the course a lot of observations were made. We will restrict ourselves to examples of some interesting phenomena, concerning general behaviour of the students, problems of working with the special kind of metaphors and problems due to lacking knowledge of statistics. We will thereafter give one example of the introduction of a metaphor.

(a) Study behaviour
A number of students made a habit of not following one of the teacher's instructions, namely that they were to write down the precise text of the system commands and to work out the parameters for a few exercises using the documentation. They preferred to attempt to use the fragmentary knowledge they had at that moment in a trial and error way on the system, and did not attend the introductory discussion on the initial problems.
Nearly all students had a lot of problems with the folklore of the (very user unfriendly) system; the login procedure for instance is a continuous source of errors.
A more general problem is the convention to end a message to the system by pressing the return key. The students kept forgetting this and repeatedly were observed waiting for a reaction of the (admittedly slow) system when the system was doing the same. For these students (and for every onlooker) their intention was fully clear.

(b) Working with metaphors
The first introduction to metaphors and especially the introduction of a slave to be given precise instructions met with some misunderstanding, in that the students were not used to formulating their commands in an algorithmic way. The sorting exercise resulted in a variety of imprecise and internal inconsistent solutions. The teacher needed a lot of time giving extensive examples to clear up this problem.

(c) Knowledge of statistics
We discovered that for some students concepts in statistics that had been taught a few years ago were lost. They had forgotten about the distinction between different correlation coefficients, or were indeed not surprised when they obtained a complete correlation matrix filled with ones (1).

(d) Introduction of a metaphor - an example
The idea of a library was used as an analogy for the file handling system. The teacher started with the introduction of the concept referring to general experience: Books are stored according to the alphabetic order of the name of the authors. In his metaphor he made it clear that, in this analogy, there is one unique author for each book. The students were asked to mention the kind of manipulations that should be made possible and which facilities should be installed. The group of student produced a list of functions that were written on the blackboard:
- locate a book,
- borrow a book,
- return a book,
- insert a new book,
- remove a book,
- list all books (to which the teacher added: e.g. alphabetically or according to the date of acquisition).

There were more suggestions, e.g. "read" or "copy", but the teacher argued that these actions need only be performed after a book has been borrowed. Since in that case the book is already no longer in the library's files this actions were left out of the metaphor.
The teacher now argued that the same kind of actions may be performed on files (analogue to books): these may be located, taken out, added, removed, put back (after something has been changed to the file - different from books in a library) and listed. A "library" of files in this system is called a master-file.

For a precise description of the semantics and syntax the students were referred to the user manual. They were presented some exercises designed to let them find out the most common instances of file handling, looking up the commands in the manual, writing down their trial solutions, trying them out on the terminal and discussing the errors in the group.
Later in the course, when some problems arose about the concepts file and masterfile, the teacher reminded the students of the difference: "Your masterfile is your library, The files are the books, and books you may read". This immediately solved the problem.
In the second week the concept systemfile was introduced, which is a central concept in SPSS. In that case the metaphor is no longer valid. The teacher lacked an adequate metaphor, and we think he did not succeed in solving all problems in the available time.

2.2.2. description of the variables

(a) computer literacy and awareness
The first variables we took into consideration were those we described in 2.1.2. (a): In average the subjects had a background of less than one day of computer experience (CASTx) and less than one day of computer education (CASTed). Because of this the subjects can be rated "novices" in our opinion.

(b) cognitive style
The dimension we used as an example for this domain, impulsivity-reflexivity, showed a nearly "normal" distribution: the scores showed a reasonable symmetric variation, there were no "outliers" nor did we detect any hint to bi-modality.

(c) metaphors
The variables we used in our analysis were derived from the questionnaires illustrated in 2.1.4. For each measure we developed some absolute scoring criterion (e.g. for the video-apparatus metaphor: the number of correct functions mentioned as compared to a canonical list, subtracted by the number of irrelevant or redundant functions). The data were scored separately by two different judges, in order to establish the reliability of the scoring method. The reliability coefficients turned out to be in between .89 and .96 (for the video metaphor) for the different variables, which was satisfactory for out purposes (with one exception, the dish washer: .80, which points to scorings rules that are insufficiently precise). For the pre-test questionnaires the scores showed a pronounced central tendency, and a symmetrical distribution. At the post-test the computer model (introduced with the metaphor of the dish washer) showed a clear bi-modal distribution: some students had a clear and consistent image of the system whereas others did not show even a fragmentary relevant model. For the questions concerning the editing job all subjects scored conspicuously low in the post-test. The notoriously bad quality of the system that had to be used in this respect surely is the cause for this phenomenon. The sorting problems and the questions about file manipulation (analogous to the video metaphor) showed a nearly "normal" score distribution at the post-test.

(d) Statistical computing
These variables are already described in section 2.1.4 (b). The correlation between pre-test and post-test were not significant. We found a considerable, statistically significant difference between the scores on the two (nearly identical) tests. A clear improvement in the semantic aspects of statistical computing may be concluded.

2.2.3. Relations between the variables

In this section we only deal with those variables that might contribute to a better understanding of the model of the system in the minds of the subjects. The relation we expected to find between the metaphor variables and their pendants we measured after the course was completed, was not apparent: we didn't find any significant statistical relation between the variables.

2.3. Interpretation

We assumed that having a more or less adequate mental model, measured via a metaphor that was going to be used during the course itself, would facilitate the learning of the complex material presented in the course. There are two possible explanations for the absence of any relation:

(a) In retrospect we note the content of the pre-test, the metaphors, was of a rather semantic nature, whereas the post-test asked for the syntactic structure of the system commands. We are in fact measuring conceptual notions that are located at two different levels in Moran's representation of the human computer interface (Moran, 1981).
(b) The course itself was of such a quality and was taught so well that every subject mastered the concepts and models of the system totally.

To check for explanation (b) we took a second look at the results of the post-tests regarding the models of the system. This learned us that we could discard possibility (b). The notorious user unfriendliness of this system did confuse the subjects so much that they couldn't perform any better. So, for a possible explanation of the absence of relations we assume that explanation (a) is feasible. This means that pre-test and post-test are incompatible as they are oriented towards different levels of human computer communication. In replications of this study one should avoid such a confusion.

Another relation not confirmed in our data was between cognitive style (operationalised in our test on impulsivity/reflexivity), and the introduction of novices to a computer system. A possible explanation is that this measure will only show effects in very short courses (e.g. one day). When subjects are able to practise during several days, the effect of being impulsive or reflexive can readily disappear.

We didn't find a relation between computer awareness and pre/post-tests on mental models. This might be explained by the fact that computer awareness is measured at such a basic level that after having followed a course like the one described here, all individual differences have disappeared.

3. EVALUATION OF THE METHOD

Although the results in this pilot study are negative in terms of the hypothetical relations we investigated, the positive effect is the illustration of a method to evaluate relations between novices' characteristics, strategy of the course and the resulting mental model of a system. The possibility is demonstrated to establish the relation between these variables in a non experimental situation, as will be found in normal computer education. The results of research in this domain will have to be accumulated by many replications of this kind of studies, with variation in the three groups of variables we mentioned. "Negative" results as well as "positive" ones help to reveil the complex relation in this field.

REFERENCES

Anderson, R.E., Hansen, T.P., Johnson, D.C., Klassen, D.L. (1979). Minnesota Computer Literacy and Awareness Assessment. Minnesota Educational Computing Consortium, St. Paul.

Cooper, M.F. (1982). Introducing SPSS (7 video tapes, course notes and SPSS exercises). NUMAC, University of Newcastle upon Tyne.

Eason, K. (1983). Metnodological issues in the study of human factors in teleinformatic systems. Behaviour and Information Technology, 2, 357-364.

Kagan, J., Rosman, B.L., Day, D., Albert, J., Phillips, W. (1964). Information processing in the child: Significance of analytic and reflective attitudes. Psychological Monographs, 78.

Kahney, H. (1983). Problem solving by novice programmers. The Psychology of Computer Use. T.R.G. Green, S.J. Payne, G.C. van der Veer (eds). Academic Press, London.

Lawson, H.W. (1982). Understanding Computer Systems. Computer Science Press, Rockvill, MD.

McKeithen, K.B., Reitman, J.S., Rueter, H.H. and Hirtle, S.C. (1981). Knowledge Organization and Skill Differences in Computer Programmers. Cognitive Psychology, 13, 307-325.

Moran, T.P. (1981). The Command Language Grammer: a representation for the user interface of interactive computer systems. International Journal of Man-Machine Studies, 15, 3-50.

van Muylwijk, B., van der Veer, G.C. and Waern, Y. (1983). On the implications of user variability in open systems - An overview of the little we know and of the lot we have to find out. Behaviour and Information Technology, 2, 313-326.

Nie, N.H., Hull, C.H., Jenkins, J.G., Steinbrenner, K., Bent, D.H. (1975). Statistical Package for the Social Sciences. Mc.Graw-Hill, New York.

Norman, D.A. (1983). Some observations on mental models. Mental Models. D. Gentner, A. Stevens (eds). Lawrence Erlbaum, Hillsdale.

Peelle, H.A. (1983). Computer Metaphors: Approaches to computer literacy for educators. Computers and Education, 7, 91-99.

van der Veer, G.C. and van de Wolde, G.J.E. (1983). Individual differences and aspects of control flow notation. The Psychology of Computer Use. T.R.G. Green. S.J. Payne, G.C. van der Veer (eds). Academic Press, London.

COGNITIVE ASPECTS

HUMAN COGNITION AND HUMAN COMPUTER INTERACTION

Werner Schneider, Mats Lind, Robert Allard, Bengt Sandblad

Uppsala University
Sweden

Not too long ago the programmer and the program user were one and the same person. Since the user had created the program he knew what to expect from it, what it could and couldn't do, how to control and change it etc. The diffusion of computers in our society and their usage in an ever increasing number of professional fields has changed this picture. In most cases the user is no longer the person who has written the program in question. This separation causes a considerable number of problems concerning the cognitive ergonomics of human computer interaction. It is the purpose of this paper to define some of these important problems and to indicate a way of solving these problems. Other important aspects of human computer interaction such as physical and social factors will not be considered.

1. THE COMPUTER AS A PARTNER IN A CONVERSATION.

The problems a user faces when confronted with a new computer program very much resemble the problems a person faces when confronted with another, unknown, person. There are certain things "in the head" of the other person - ideas, intentions, knowledge etc. that cannot be directly perceived, and must therefore be induced through communication with that person. Similarly, when confronted with a new program, a user cannot know what is inside the computer, "behind the VDU screen", because invariably it looks the same as every other system he knows. The way in which the program works and the possibilities it offers have to be induced from communicating with the program and from additional information sources such as manuals, previous users etc. The similarity of these two situations have led to attempts to add a "natural language interface" to programs, since person to person communication is often successfully achieved through natural language. Much of the research concerning natural languages and their implementation in computer programmes has dealt with intralinguistic problems such as syntax, semantics, word ambiguity and so on. We would like to indicate a few components of person to person communication that are of equal importance, thus warranting consideration in the present context. Their importance is so great that even if the intralinguistic problems were solved, a better HCI (Human Computer Interface) could not be expected.

Although it is true that human beings are unique they also share a set of properties. Some of these properties play a great role in human communication. It is not our intention to try to make an exhaustive list of these or to structure what person to person communication really is, but instead to point out a few important aspects that have a direct bearing on human computer interaction.

To begin with, human beings have a very elaborate perceptual system with an enormous capability. The computational power needed in the real-time processing of a complete three dimensional representation of the environment in full colour is, to say the least, impressive. These perceptual capabilities underlie at least three aspects of communication:

(a) Since both participating partners can perceive the surrounding environment, language can easily and without loss of information (probably rather a gain) refer to things or events for which the participants in the communication do not have clear concepts or words. For instance, "Look what is happening over there, I would like to be able to do that", or "Give me one of these, please".
(b) Communication can be totally unambiguous although there is an intralanguage ambiguity. "Watch out for the mustang" is perfectly clear on a freeway and in a corral, although the information transferred is quite different (Fitter, 1979). This also means that communication can be short and powerful since not so many explanations (= resolutions of ambiguity) have to be given.
(c) There is a whole system of communication not using words but intonation, body posture, gestures etc.

On top of these perceptual capabilities human beings also share another important property - human beings are adaptive. This means that we can adapt to each others way of thinking and reacting to enhance communication. Just think of the number of instructions a new apprentice needs in an unfamiliar work site and compare this to the number of instructions he would need after having worked there for a long time.

Seen in this light the computer capacities needed for the efficient use of person to person communication as a model for human computer interaction are far greater than natural language interfaces can offer. It is therefore our firm belief that we ought to look in another direction in the coming years to overcome the problems encountered in human computer interaction.

2. A DIFFERENT APPROACH TO HCI CONSTRUCTION

Up to this point we have been concerned with those human abilities which computers do not possess. The power of computers is however of another nature. The opportunity to do number crunching and to control visual display devices is something which only computers can offer, and which has increased the number of ways in which problems can be solved: We are no longer restricted to modelling person to person communication. The problems previously discussed are caused by the fact that the computer programs are no longer only used by those who have written them. The normal situation is now such that people are confronted with new programs without knowing how to run them, what they can do, how the processes can be controlled and so on. Instead of seeing the program as a "black box" that can only be accessed through a language interface, a solution would be to present the program in such a way that the user can make a working model of it as easily as possible. Before describing the way in which this can be achieved we will briefly discuss what we consider to be the nature of mental models and their information.

2.1. The nature of mental models

Rasmussen (1980) gives an excellent account of what we know of human cognitive functioning and distinguishes between a "high capacity, parallel processing system which functions subconsciously, and a sequential conscious processor of limited capacity" (p. 72). What we would like to suggest is that the two systems are dependent of one another. The high capacity, parallel processing system is the basic one which allows us to move and orient ourselves in our environment. It is therefore primarily concerned with constructing and using models of our environment based on spatio- temporal relationships. The operators and parameters of these models formed by the high capacity, parallel processing system, we suggest, are the basic entities of the low capacity sequential system. This would mean that if the high capacity parallel processing system has not been allowed to develop a relevant model of a specific environment, the low capacity, sequential system cannot function in a relevant way.

If the entities with which the low capacity, sequential system works are derived

from the dynamic world model suggested above, the construction of HCI's will be greatly influenced: every part of a program that can be represented as a "time space pattern" (Rasmussen) in the HCI, will be represented as such. This is true for every type of program, even for programs in which control in terms of high level concepts is required, not for the purpose of control, but to enable the user to form a correct opinion of what these higher order concepts actually refer to. If this is not done, the user will only be able to understand the program to the extent that his already existing world models can form a basis for the understanding of the concepts used in the HCI.

The goal in the construction of HCI's is therefore to represent as much of the program and its functioning as possible as time space patterns.

The term "time space patterns" needs further specification. Since our memory for spatial relations is extremely efficient we can retrieve both objects and information on the mere basis of their physical location: every concept which is represented in the HCI as a spatial pattern must also have a physical location, i.e. a spatial relation to every other defined pattern. This seems to be one of the most important features in the formation of mental dynamic world models - being able to keep track of things, not by their name, but by their physical location.

3. IMPLICATIONS FOR THE CONSTRUCTION OF HCI'S

The representation of the program should be such that it minimises the effort needed by the user to form a correct model of the program. A program can be considered to be a collection of OPERATORS that can act on a set of DATA.

At a specific instance of time the configuration of data can be called a STATE.

Considering the nature of mental models we can conclude that both the states and the operators should, to whichever extent possible, be represented in such a way that they are perceivable. Since our primary source of information about spatial relationship is vision this means that states and operators should be represented by position, colour, form etc. This does not mean that verbal labels should be abandoned totally, on the contrary, they supply useful complementary information in much the same way as a text accompanying a picture can highlight certain aspects and lead to a quicker and better interpretation of new visual information. Indeed some of the most common computer applications involve words as the basic data type, in which case the representation of the states is naturally in terms of patterns of words. The operators do not, however, have to be of a verbal nature.

3.1 States

The level of detail of the representation of the states is determined by both the set of goals a user can have when interacting with the program and by the set of operators available.

Since adults have a tendency to perceptually group visually presented information according to the so called Gestalt principles (similarity, proximity, good continuation etc.) information that is logically connected should be represented in such a way that it is also perceptually connected.

This grouping should be done taking more than one dimension into account because of the increasing ease of discrimination caused by the presence of redundant information. Examples of such aspects are place, form and colour. Care must be taken to make contradictory grouping impossible.

If the data are of verbal nature and the task of the operator is to read and

evaluate such material all the information needed in an evaluation should be presented simultaneously because of the limited performance of the human memory in tasks involving coded material.

If these data are to form the basis of a searching process for certain, to the designer unknown, pieces of information, there must be a possibility to display large amounts of data simultaneously for the same reason as above.

3.2. Operators

The greatest difficulty in forming mental models of a program possibly does not reside in grasping the data involved and their interpretation, but to know and to understand the set of operators available and the way in which they work. This process can be said to have four components, discriminating/grouping the operators, remembering this, grasping their effect on the data and remembering this. From memory research we know that humans have been shown to have a remarkable memory capacity during recognition tasks, but have had less remarkable results in recall tasks. This together with our discussion above about the spatial nature of memory and also of the spatial grouping tendencies implies:

The representations should always be presented at the same physical location, i.e. to allow the user to form a spatial representation of the place where the operators are to be found. The representations of the operators should be grouped with respect to more than one aspect according to the principles outlined for the data above.

The form of these representations should be related to their function.

3.3. Relation to other attempts

Some of the steps that have been taken in the direction of constructing a HCI which concretely represents the programs, such as LISA by Apple and STAR by Xerox, do not fulfill many of the important requirements mentioned above. For example, the spatial relations between the operators are often undefined, the structuring of the representation is not always done according to the principles described here etc. The use of a small screen in this kind of HCI also necessarily implies that sequential restrictions are imposed on the user prohibiting efficient use of the programmes.

4. REALITY

Our everyday environment usually fulfills all the requirements stated above (unless we have a computer of course). These environments have to a large extent been constructed by man, but the materials used, and the hundreds of years of evaluation and invention, have made them easy to work with.

An ordinary office for instance, resembles the environment in which an economic/administrative program is used on a computer in many ways. The way in which the task of information retrieval is performed can be based on the principles of recognition, spatial position and perceptual grouping. Paper itself takes up a certain amount of space. Bookshelves are filled with paper (spatial positions) that is usually grouped in some way or another ("accounting on the top shelf" or specially coloured markers). To find a specific piece of information you only have to remember a very approximate location (not a specific name or code), to look in that direction and to let your enormous capacity for visual scanning and recognition work and soon you will have found what you were looking for. Please note how such a process can be disturbed (not completely disabled) by several factors. First, if the papers were continuously reshuffled by some other person spatial location as retrieval

information could not be used any more. Second, if the papers concerning a specific topic were spread all over the office in different sized files or volumes it would also be harder because grouping ("chunking") could not be done. Not to speak of the situation which would arise if you did not have any access to your office at all but instead had to ask for all the information you wanted with only a serial number.

Some of the operators available to you are abstract and can only be remembered by recall, others do have a concrete representation in terms of tools such as scissors, adhesive tape, stapler etc. All of these representations have a place, form and colour. Hence you can find them in the same way as has been described above. On top of that their shape is a good indicator of their function.

5. CONCLUSIONS

In this paper we have tried to indicate some of the enormous difficulties involved in trying to simulate a human being in order to construct a computer program which is easy to use, even for beginners. We have also tried to describe a very different approach, in which the computer tries to simulate those important aspects of the environment which our sensory, motor and intellectual abilities have been shaped to handle.

A number of implementations and experiments have been started according to these guidelines and we will present some of the results at the conference.

REFERENCES

Fitter, M.(1979). Towards more "natural" interactive systems. International Journal of Man-Machine Studies, 11, 339-350.

Rasmussen, J.(1980). The human as a systems component. Human Interaction with Computers. Smith, H,T. & Green, T.R.G. (Eds.). Academic Press, London.

UNDERSTANDING COMPLEX DESCRIPTIONS

Collin Potts

Department of Computing
Imperial College of Science and Technology, London
UK.

User-system interaction is starting to receive the attention it deserves. A picture is starting to emerge from work in cognitive ergonomics of the requirements for user interfaces, for example for text editors (Card et al., 1983), command languages (Barnard et al., 1981) and programming language syntax (Green, 1980). In a short sketch of the research opportunities that exist, Shneiderman (1982) lists several similar issues, such as menu selection, on-line assistance, etc. He also draws attention to issues such as overcoming anxiety and fear of computer usage. I shall not comment on such social and organisational issues, but I want to suggest that attention to ergonomic issues of the first type neglects a range of important human factors in system use. These operate at a conceptual level beneath the external (e.g. graphical or pretty-printed) characteristics of the interface. The problem I want to address relates to these. It is this: how can an information system be designed so that the complex descriptions it supports and which are manipulated by the users can be rendered into forms that are harmonious with the users' mental models?

In the first part of the paper I shall discuss a conceptual architecture for information systems which emphasises the understandability of the descriptions they embody. The architecture consists of an internal description which may be manifested by applying two classes of transformation, forming separate conceptual and external interfaces. In connection with the conceptual interface, I discuss the relevance of current work in cognitive science, especially text linguistics. I shall ignore the external interface, as this has received the most attention in the cognitive ergonomics literature.

In the second part of the paper, I shall concentrate on a class of complex descriptions of particular relevance to software engineering, specifications. A specification is usually thought of as a text and a specification language as a language. A specification, however, is simply a description of a reality that does not exist yet and a programming support environment is, amongst other things, an information system for manipulating such descriptions. Thus in addition to the effects of syntactic and physical factors (e.g. layout) on the understandability of specification, we should also be examining the conceptual models required of the specifier by different specification language.

1. INFORMATION AND DESCRIPTIONS

Much of our interaction with computers consists of manipulating complex descriptions, and the understandability of information systems should be seen in this light. To understand such a system one must understand what is being described and how the description describes it. In other words, the user must understand the domain of discourse itself (but not necessarily all of it in detail, and possibly with on-line help from the information system), and the information system must present the description it contains, and the parts of that description that should

be brought to bear on the solution of application problems, in away that is consonant with that understanding.

1.1. Views and the canonical formalism

Because a system usually caters for a variety of categories of user, its description of the real world must be capable of being manifested in a variety of ways. In addition to this static diversity of viewpoints, a user may wear a range of different organisational 'hats', and may use the system on different occasions to fulfil functional or organisational roles.

This notion of 'views' is familiar from the database world. One way of supporting views is to maintain separate descriptions with rules effecting the transformation between them (Figure 1a). This approach has several disadvantages:

(a) it is space inefficient (and inelegant);
(b) it is static;
(c) the descriptions are bound to become inconsistent eventually.

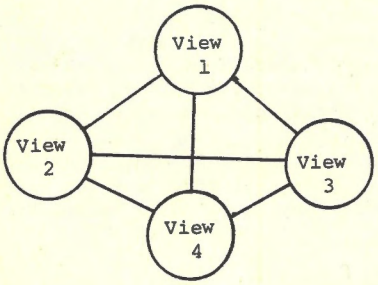
An alternative is to select one of the descriptions and adopt the formalism or data model in which it is expressed as canonical. Better still, a canonical formalism can be devised which is separate from any of the conceptual views, but which has desirable formal properties, and the users' views can all be derived from it (Figure 1b). Thus a description expressed in the canonical formalism may be quite unpalatable as an external view, but need never be inspected or manipulated by anyone in this form.

This two-level model requires further enhancement. It has no distinction between the steps which reorganise and simplify the description in the canonical formalism, and those which take the resulting conceptual description and present it in different external formats (e.g. by listing tables, or drawing graphs). The complete architecture is depicted in Figure 2.

The internal description is mediated through two interfaces, the conceptual interface, producing a conceptual description, and the external interface, producing physical output. The process of generating output from the internal description is called manifestation and the inverse process, of encoding descriptions is called formulation

The advantage of separating conceptual and external levels, is that the external interface comprises what is usually called the 'user interface'. Wasserman (1981) and Edmonds (1981) demonstrate that this can be detached from the functionality of the system and implemented separately. This is not possible at the conceptual level, which is intertwined with the function of the system and the description of reality that it embodies.

(a) Views as autonomous descriptions



(b) Views as manifestations of an internal description in a canonical formalism

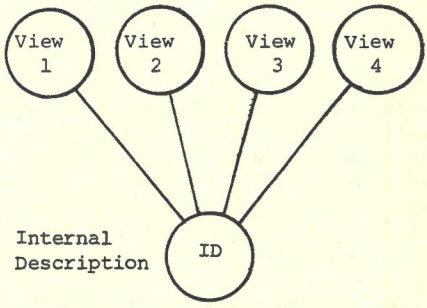


Figure 1 Alternative methods of representing multiple views

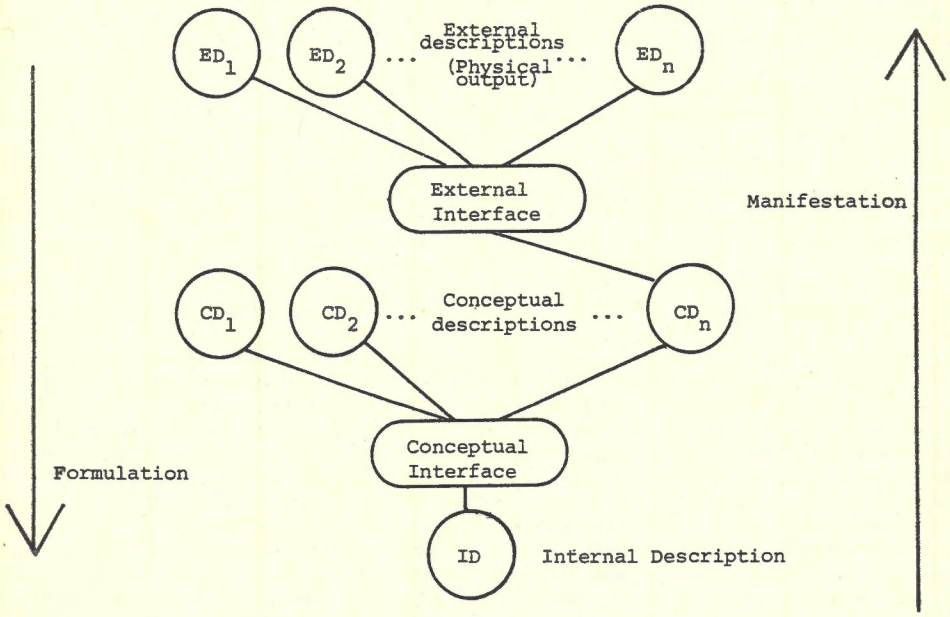


Figure 2 Conceptual architecture for information systems

1.2. Manifestation

Extracting specific information from the internal description, particularly

information that would be represented differently in different views, in rather like summarising the gist of a passage. Van Dijk (1978) has suggested that summarising natural language depends on a set of 'macro-operations', which transform the propositional content of the text into a propositional structure at a higher level, or 'macro-structure'. Domain-specific knowledge structures determine when macro-operations are appropriate.

Good psychological evidence has been obtained that summarisation and memory for prose depends upon; a local propositional representation, a small working memory, and the construction of a macro-structure from macro-operations. For example, Kintsch and Van Dijk (1978) and Vipond (1980) obtained a close fit between subjects' recall of passages and the simulated recalls based upon the model. Spilich et al., (1979) investigated the knowledge structures required to bring about macro-operations during comprehension of baseball commentaries by baseball fans and students with little knowledge of the game. Using an analytic goal structure for baseball, they showed that fans had better recall because they were more discerning about the importance of information in the commentary.

To make complex descriptions easier to understand, therefore, it seems natural to structure the process of conceptual manifestation around operations resembling macro-operations. The steps I shall describe are similar to Van Dijk's. These are: re-formulation, filtering, generalisation and integration.

(a) Re-formulation of a description is equivalence-preserving. That is no information is lost or added, the existing description is merely re-organised so that constructs which were implicit, or 'spread' out, in the original description are brought into focus. Consider a 'produces' and 'consumes' database where agents (they might be warehouses, chemical plants or factories) produce and consume things (which might be widgets, chemicals or data flows). An agent-based description of the form:

```
a produces D
b consumes D
b produces E
c consumes E
```

could be re-formulated so that the focus is on the things being produced and consumed:

```
D goes from a to b
E goes from b to c
```

(b) Filtering deletes information which is not important to the current view, or which can be assumed by the user. Most simple database queries (e.g. "where do all the widgets go?") are filters. 'Views' in existing relational databases consist of static filters (or 'blinkers'), so that each user can see only part of the database, his or her schema being a sub-schema of the whole. Current database systems do not support views derived by operations other than filtering.

(c) Generalisation implies the need for partial ordering of types (e.g. a class hierarchy with strict property inheritance). For example, the query "which people are working on this project?" may involve a generalisation from 'senior consultants', 'programmers' and 'documentation assistants' to the superordinate class 'people'. Note that in the view in question, there may be no distinction between people: the user may not be aware that generalisation is necessary.
A special case of generalisation which is very important in interaction with information systems, but less so in summarising prose, is quantification. Quantifying the number of penurious academics who earn more than me involves an implicit generalisation from penurious academics A, B and C to the general class of 'penurious academics'.

(d) Integration is a more sophisticated form of summarisation. Domain-specific knowledge is represented in the form of attributes and associations particular to objects of a given class. Additional relations, such as time-ordering of

actions, may be represented as cues for causality. To use Van Dijk's (1978) example, the following facts:

I bought wood, stones and concrete
I laid foundations
I erected walls
I made a roof

can be integrated as:

I built a house

1.3. Knowledge structures

The above examples of conceptual manifestation steps are simple. Most real examples would consist of the composition of many such steps. The selection of the operation and their sequencing (composition is not, in general, commutative) requires intercession by knowledge structures of the kinds indicated in Figure 3.

The data dictionary is a meta-level description of the canonical formalism. It defines the data model that is instantiated by the internal description. In current implementations it would consist of the physical and conceptual schemas, a conventional data dictionary and a set of integrity checking procedures.

It is likely that the set of conceptual manifestation operations (CMO) would be supplemented by a set of composed operation 'packages'. The simplest user model would then be a definition of valid views for each user category. These views would be mapped onto the appropriate packages in CMO.

A domain model must specify a type lattice for objects within the domain of discourse. This is used primarily for generalisation operations. Attributes and associations particular to a type should also be represented. Two approaches to domain modelling are discussed by MacLean et al. (1983) and by Bartlett et al. (1984).

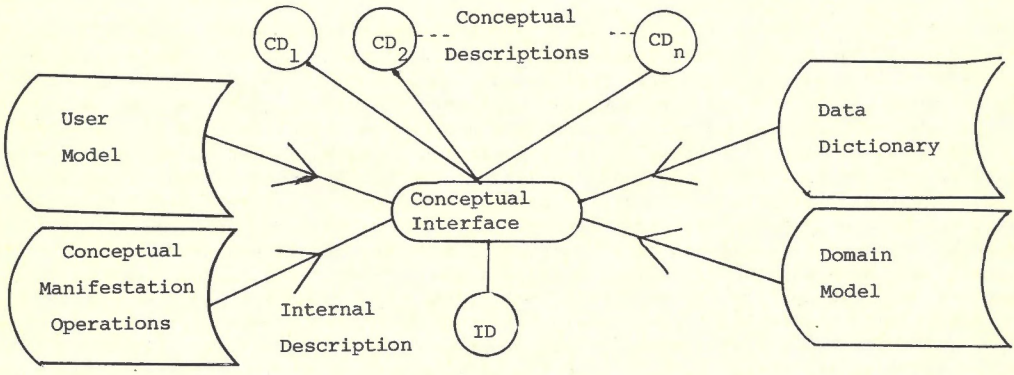


Figure 3 Knowledge sources for conceptual manifestation

2. UNDERSTANDING SPECIFICATIONS

A specification is a description of a world that has not been realised yet. For example, a software developer's specification of program function is a description of the behaviour of an imaginary program. The extra element of futurism, or intention, does not alter things as far as we are concerned; specifications are just a special kind of description, and the understandability of specifications depends on precisely the same categories of factors and macro-operations as the understandability of other descriptions.

First of all, what is a specification? The dichotomy between specification and program is an oversimplification. As Lehman et al. (1983) have shown, the software development process can be seen as the repeated application of transformations between representations. Each representation is a 'specification' with respect to the ones following. Indeed, if you were a compiler you would regard the program itself as a specification. I shall refer to all such representations of the final product, including the program text itself, as 'specifications', but I shall restrict most of my attention to specifications earlier than the program, which are more declarative.

2.1. Specifications as texts

Specifications must be understood by people, as well as being amenable to formal manipulations. Many 'specification languages' have been proposed -- see Staunstrup (1982) for an introduction -- but little attention has been paid to the understandability of specifications written in such languages, and the trade-off between rigour of specifications and the wider acceptability and usability of formalisms for expressing specifications has yet to receive much attention.

Let us first examine the notion of a programming 'language' and then look at specifications. Programmers spend a great deal of time poring over program listings. When most people think of programs they think of texts printed on green and white striped paper with sprocket holes down the side. We now have work stations with bit-mapped displays and mice (well, some of us do), but this should not disguise the fundamentally textual nature of existing programming languages. Indeed, with the current trend for high-resolution iconography of such familiar everyday objects as litter bins and filing cabinets, it is surely only a question of time before someone markets a text editor that supplies on-line stripes and simulated sprocket holes for us.

Some programming language features exist only because of the linear nature of program texts. For example, the insistence on data declaration before use is taken to extremes in languages such as Ada. Notice that the idea of a sequential scan, and therefore the linearity of language, is built into the notion of a compiler 'pass'. Now, there are strong prima facie reasons for keeping the linear notion of programming languages. Amongst other things, a program is a specification of a temporal ordering of computational events. This ordering need not be represented explicitly (e.g. in temporal logic or transition networks) because it can be derived from the text. But as one moves further back through the programming process the concepts expressed become less temporal and procedural, and more static and declarative. Yet there is a common attitude that specifications are in some sense 'like' programs and that as programming becomes an increasingly formalised engineering profession that its professionals (usually called 'analysts' in this context) will spend a great deal of their time poring over listings in much the same way as programmers do today. The only difference will be that the languages in which the texts will be expressed will be specification languages, rather than programming languages.

In fact, the word 'language' is used in two senses; as a means of communication (that is a carrier of descriptions), and as a formal system in which the adjacency and sequential ordering of symbols carries special information. The second sense implies linearity.

2.2. Specifications as descriptions

An alternative view of specifications is that they are descriptive models. This is an especially fruitful view of earlier specifications; one would seldom query a database or a specification by asking the question "tell me everything you contain", but this is precisely what a listing is. Similarly, one would not elevate to the status of a 'specification language' a database input or query language, although the specification language PSL (Teichroew and Hershey, 1977) is little more.

Of course, large programs are broken down into components, so it is not quite true to say that one must list an entire program in one go. But there is only one morphology of a program, its modular structure and this is directly reflected in the linear appearance of program texts. This makes it difficult to list aspects of the program which cut across this particular structuring, for example when one wants to know where a given variable is used (rather than where it is declared), or how the control flow of the program can get to a certain point (rather than where it can go next).

In the case of specifications, this need for multiple views is much greater. If a specification is seen as a description, rather than as a text, it can be modularised non-linearly and dynamically.

We are back to the distinction drawn in the context of information systems, between the internal description in the canonical formalism and its conceptual manifestations: thus a distinction must be drawn between the unique specification in a canonical formalism, and the set of transient external manifestations derived from it. Only some of these need be 'linguistic' in a conventional sense; others may be graphical, tabular, or even animated.

Next we shall look at a concrete example of a canonical formalism, Prolog, and how usability considerations affect the formulation of a specification even at the internal level. Linguistic manifestations will then be discussed, and finally animated manifestations.

2.3. Prolog as a canonical formalism for specifications

We have been basing a canonical structure for specifications on Prolog (MacLean et al., 1983). Prolog is very suitable as a formal representation medium for specifications because it has a formal basis in the Horn clause representation of first-order predicate logic (Kowalski, 1979) and much current research is attempting to extend logic programming to other (e.g. temporal) logics. The granularity of a Prolog specification is typically very high, consisting of a large number of uniformly small facts and rules. This makes it suitable as a canonical formalism for specifications. Prolog is a programming language and thus is directly executable. It can also be regarded as a database tool with an inference mechanism; indeed, there is no formal distinction between running a Prolog program, and querying a Prolog database. Prolog can take Prolog clauses as its object,and this permits metalevel rules (e.g. for consistency checking and for conceptual manifestation) to be expressed cogently.

Prolog, however, has several undesirable features that make it an unpleasant tool for communication at the external level. For example, there is no explicit typing of objects, which we have seen is the basis of generalisation. The granularity of the information in rules and facts is frequently far too high, so that a Prolog specification is modular in only a very weak sense.

We have examined the potential of Prolog for supporting the specification of systems using the CORE requirement analysis method (Mullery, 1979). In order to impose some discipline on the processes of information acquisition from the client, representation of CORE-specific abstractions, and writing manifestation rules, we devised a canonical formalism which consisted of Prolog plus some 'keywords', that is CORE-specific predicates (MacLean et al., 1983).

The likely use of the information does influence how that information should be represented in the canonical formalism. For example, in the case of a simple lift system the developer may wish to know the number of buttons that call the lift to travel upwards. The following formulations in Prolog are equivalent:

```
number_of(up_button, X) if number_of(middle_floor, Y) &
                           number_of(bottom_floor, Z) &
                           X is Y+Z.

number_of(up_button, X) if number_of(floor, Y) &
                           number_of(top_floor, Z) &
                           X is Y-Z.
```

The second formulation is preferable because it is a qualification of the natural, but incorrect, assumption that there is an up button on every floor. Of course, 'Z' could be replaced by the constant '1' in both formulations, but this 'optimisation' would obscure, not clarify, the relation between the cardinalities. This becomes important if an expert system explanation tool such as APES (Hammond 1982) is provided to walk the specifier through a complex chain of inferences.

2.4. Conceptual interface for specifications

The conceptual manifestation operations were introduced in Section 1.2, above. I shall now examine how they can be made useful in the context of specifications.

2.4.1. Re-formulation

An exciting area for future research is devising re-formulation rules which map a highly granular logic-based internal description into a structured object-based description based on formalisms such ad DL (Winograd 1983). For example, consider the Prolog and DL descriptions in Figure 4 taken from an imaginary specification for a university admissions system. Ignore the external layout of the descriptions, and notice instead that the DL description is focussed on the object 'course'. The problem with object-based languages like DL as specification languages is that only one view is possible (why, for example, should 'takes_course' be associated with 'course' rather than 'student'?) But if an object-based description is derived dynamically from an internal description, alternative views can be maintained without redundancy.

2.4.2. Filtering

Filtering is clearly a major requirement for understanding specifications. The simplest form is filtering by gross modularisation (e.g. by listing a single procedure from a large program), but this is inflexible. Fischer and Schneider (1984) have developed tools that allow Lisp programmers to make program fragments invisible in the displayed manifestation of the code. The fragments are frames and their roles (e.g. a function's callers, documented purpose and code), and a filter is defined by controlling the visibility of each of the roles.

(a) Object-based DL description (after Winograd 1983).

```
course
     roles:
          students: SET OF student
          maximum_enrolment: integer
          enrolment (COMPUTED): integer {SIZE OF students}
     predicates:
          a course is full if
               enrolment = maximum_enrolment
          a student is taking a course if
               student MEMBER OF students
     categories:
          level: {elementary, advanced}
```

(b) Logic-based description with typing:

```
valid_students(x:course, y:students) if students(x, y).
valid_maximum_enrolment(x:course y: integer) if
     maximum_enrolment(x, y).
valid_enrolment(x: course, y: integer) if
     enrolment(x, y).
enrolment(x: course, y: integer) if
     students(x, z: students) &
     SIZE_OF(z, y).
full(x: course) if  maximum_enrolment(x, y:integer) &
                    enrolment(x, y).
take_course(x: student, y: course) if
     students(y, z: students) &
     MEMBER_OF(x, z).
valid_category(course, level, elementary).
valid_category(course, level, advanced).
```

Figure 4 Illustration of re-formulation: Logic-based internal description and object-based manifestation

2.4.3. Generalisation and particularisation

Generalisation, and its inverse particularisation, are important in specifications where types can inherit properties from supertypes. In the above example, a course may be elementary or advanced. The specifier must be able to inspect the properties of elementary and advanced courses (they are the same as those of 'course' in this simple example) even though they are not described explicitly, but must be inferred through particularisation operations.

During the course of specifying a system, the specifier may wish to quantify instances on the basis of generalisation. For instance, the number of buttons required in a lift system (not a simple question - think of a lift system you know well) should be inferred from the numbers of various sorts of buttons (the ones on each floor, the ones in each lift, etc.)

2.4.4. Integration

Perhaps the clearest need for integration operations is not for specifications themselves, but in summarising traces from running programs, or scenarios used to exercise specifications. An example of a simple event grammar for summarising traces from distributed systems is given by Bates and Wileden (1982), in which events can

be composed into 'macro-events' by regular expressions.

2.5. Validation of specifications.

The examples of manifestation given above are all static, and the categories have been derived from text linguistic work. There is another way of developing an understanding of a specification which is very different. This is validation by execution.

Validation answers the question whether the current specification represents what the client wants and really needs (MacLean et al., 1984). Inspection of a suitable static manifestation of the specification may suffice. In addition to looking at diagrams or formatted texts, static inspection may include theorem proving. Nevertheless, computer systems are unique among engineering and information artifacts in the primacy of their behaviour. To specify a system, one specifies what it does. Thus to validate specification one needs to be able to generate a model of the system's behaviour. To achieve an understanding of the complex behavioural consequences of a specification it is often necessary to convey the flavour of using (or being) the system and walking through its behaviour directly.

The current interest in rapid prototyping stems from the need to validate specifications by direct observation of behaviour. Rapid prototypes however suffer from several inadequacies: they are derived informally so the accuracy of their portrayal of behaviour is not guaranteed; there is no formal means of feeding back conclusions about the behaviour of the prototype to the specification; and it is tempting to encourage the clients (who may not be end-users) to 'play' with the prototype, rather than experiment carefully (indeed, a prototyping effort can easily become a public relations exercise).

Within the framework described in this paper, my colleagues and I have experimented with two animation techniques for specifications which are more rigorous than rapid prototyping. In one, a specification of a geochemical computer-aided learning system (Potts et al., 1983) was encoded in Prolog. This could be animated using suitable support tools. In the other, simulation models (Bartlett et al., 1984) were generated from JSD specifications (Jackson, 1983).

3. UNFINISHED BUSINESS

While preliminary work has been accomplished in supporting conceptual manifestations of a specification, including executable models, separate from its canonical formalism, further work is required in the following areas.

(a) How should domain-specific knowledge, used in generalisation and integration, be represented?
(b) How can views, which depend on conceptual manifestation steps other than filtering, be defined?
(c) How can categories of users best be categorised in terms of the views they require?
(d) Given that the canonical formalism is likely to be unpalatable to people, how should formulation take place?

ACKNOWLEDGEMENTS

This research was supported by the Advanced Research Projects Agency of the Department of Defense and the Rome Air Development Center (RADC) and was monitored by the

Air Force Office of Scientific Research under contract No. F49620-82-C-0098.

Most of the ideas described in this paper have arisen from discussions with Manny Lehman, Andy Bartlett, Brian Cherrie and Roy MacLean. Matthey Morgenstern will recognise, metamorphosed, some of our database wish-list.

REFERENCES

Barnard, P., Hammond, N., Morton, J., Long, J., Clark I. (1981). Consistency and compatibility in human-computer dialogue, Int. Journal Man-Machine Studies 15, 87-134.

Bartlett, A.J., Cherrie, B.H., MacLean R.I., Potts C. (1984). The derivation of Ada executable validation models from JSD representations. Imperial College of Science and Technology, Dept. Computing, Research Report Doc 84/3.

Bates, P.C., Wileden J.C. (1982). EDL: a basis for distributed system debugging tools' Proc. 15th Hawaii Int. Conf. Sys. Sci. Software, Hardware Decision Support Systems, Special Topics. Vol. I.

Card, S.K., Moran T.P., Newell A. (1983). The Psychology of Human-Computer Interaction. Erlbaum.

Edmonds, E.A. (1981). Adaptive man-computer interfaces. Computing Skills and the User Interface. Coombs, M.J., Alty J.L. (Eds.). Academic.

Fischer, G., Schneider, M. (1984) Knowledge-based communication processes in software engineering. Proc. 7th Int. Conf. Software Eng., IEEE Press.

Green, T.R.G. (1980) Programming as a cognitive activity. Human Interaction with Computers. Smith H.T., Green T.R.G. (Eds.). Academic.

Hammond, P. (1982). APES (A Prolog Expert System Shell): a detailed description. Imperial College, Dept. Computing Research Report Doc 82/10.

Jackson, M.A. (1983). System Development. Prentice-Hall.

Kintsch, W., Van Dijk, T.A. (1978). Toward a model of text comprehension and production. Psychol. Rev. 85, 363-394.

Kowalski, R. (1979). Logic for Problem Solving. North Holland.

Lehman, M.M., Stenning, V., Turski W.M. (1983). Another look at software design methodology. Imperial College, Dept. Computing Research Report Doc 83/13.

MacLean, R.I., Potts, C., Cherrie, B.H., Bartlett, A.J. (1983). The MINEX Model and its Language. Imperial College, Dept. Computing, EMMA Technical Report TR3 (available from the author).

MacLean, R.I., Potts, C., Bartlett, A.J., Cherrie B.H. (1984). Validation in the software process. Proc. Software Process Workshop, Egham, IEEE Press.

Mullery, G. (1979). CORE: a method for controlled requirement expression. Proc. 4th Int. Conf. Software Eng., IEEE Press.

Potts, C.,MacLean, R.I., Bartlett, A.J. (1983). Modelling the domain of discourse as a precursor to requirements analysis. Imperial College of Science and Technology, Dept. Computing, Research Report Doc 83/16.

Shneiderman, B. (1982). Human factors of interactive software. Enduser Systems and their Human Factors, Blaser, A., Zoeppritz, M. (Eds.). Springer-Verlag.

Spilich, G.J., Vesonder, G.T., Chiesi, H.L., Voss, J.F. (1979). Text processing of domain-related information for individuals with high and low domain knowledge. Journal Verbal Learning and Verbal Behaviour, 18, 275-290.

Staunstrup, J. (Ed.). (1981). Program Specification. Springer-Verlag.

Teichroew, D., Hershey, E.A. (1977). PSL/PSA: A computer-aided technique for structured documentation and analysis of information systems. IEEE Trans. Software Engineering SE-3: 41-48.

Van Dijk, T.A. (1978). Text and Context: Explorations in the semantics and pragmatics of discourse. Longmans.

Vipond, D. (1980). Micro- and macro-processes in text comprehension. Journal of Verbal Learning and Verbal Behaviour, 19, 276-296.

Wasserman, A.I. (1981). Software tools in the User Software Engineering Environment. Tutorial: Software Development Environments. Wasserman A.I. (Ed.). IEEE Press.

Winograd, T. (1983). Language as a Cognitive Process. Vol. 1. Syntax'. Addison-Wesley.

DO WE REALLY HAVE CONDITIONAL STATEMENTS IN OUR BRAINS?[1]

Jean-Michel Hoc

Laboratoire de Psychologie du Travail de l'EPHE (ERA CNRS), Paris
France

1. INTRODUCTION

Programs which beginners are asked to write, more often than not, correspond to tasks that can be executed by hand. The programming strategy usually employed is one in which a well-known procedure is adapted to adhere to the rules of operation of the formal machine underlying the programming language being used (Hoc, 1983a).

However, it is not enough to simply adapt a procedure, it must be explicited as well. This requires an awareness of the control structure, in other words, data identification operations and transfers of control. This paper will examine the nature of the control structures on which this awareness is based, and not the complex mechanisms involved in becoming aware (Piaget, 1974a, b).

In procedural programming language control is expressed by means of tests. It has been shown however (Miller, 1981), that beginners find it difficult to construct those test statements in conditional structures or at the ends of iterations. Several research papers have been devoted to this question and in particular to the facilitating effects of different languages (Sime et al., 1977; Green, 1980; Van der Veer and Van de Wolde, 1983). But it is possible to go even further and to examine the effect of such tests in control structures of procedures which seem to be algorithmic and which are executed by hand. If not, this would explain one of the reasons for the difficulties experienced by beginners. This hypothesis can be considered in the light of two current areas of research in psychology:

(a) Research on attention (Richard, 1980) which has highlighted the importance of states of preparation (expectation phenomena) linked, in particular, to frequency and recency effects;
(b) Research on typicality (Cordier & Dubois, 1981) which exploits the delay in decision-making in class-sorting problems in order to show that, in the subject's representations, there are typical (short delay) and atypical (long delay) examples which are not necessarily linked to frequency effects.

If such effects exist, it can be expected that certain identifications will be omitted during the execution of the procedure, either:

(a) because the subject adapts to the frequency of events or he is sensitive to the recency effect or,
(b) because the data being processed are represented semantically in the long-term memory, implying typicality effects.

This experiment mainly examines the second phenomenon by comparing the control structures during the execution of procedures in two different situations. In the first, the subjects rely on a strongly semantic representation of the data and in

the second, on a much more abstract representation. In both situations, the data to be processed have the same "objective" structure. In the "semantic" situation we anticipate that the subjects will not be able to deal with certain identifications which could have been made in the "abstract" situation in which exhaustive search is possible.

The subjects are placed in a command situation in front of an interactive computer device. The data transformations are controlled by function keys so that the control structures can be identified by response latencies. The different representations are achieved by modifying certain characteristics of the device which affect neither the data structure nor the commands available but only the way in which the data can be accessed.

2. METHOD

2.1. Subjects and Task type

Twenty adults with no particular computer skill took part in the experiment. They were first presented with a task comparable to the task (of using a keyboard and screen device) they would have to do in the real test situation. It concerned the updating of a stock (fig. 1).

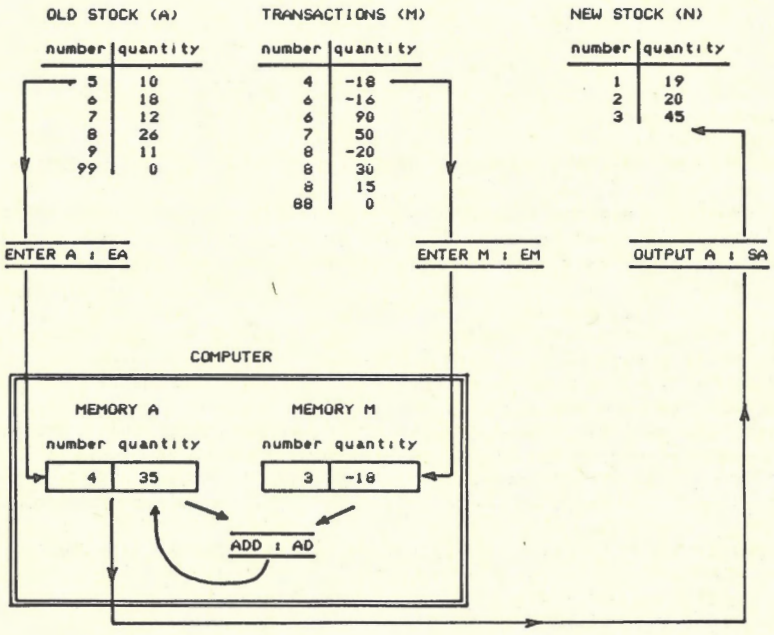


Figure 1 Task of reference

The subjects had at their disposal: the stock position the evening before (Old Stock file: AS) and the following day's transactions (Transaction file: MVT). The items

were listed according to their classification numbers - lowest first. The subjects had to construct a new file (New Stock, NS) similar to the file Old Stock (AS) but including the quantities actually in stock after the day's transactions. The subjects were then shown how to use the transformations common to the two experimental devices. They had to learn how to use the keyboard display, the system presenting data files and the processor, together known as a "computer".

The computer contains two memories, each containing one item from one of the corresponding files (memory A for the file AS and memory M for the file MVT). The subject gives the computer commands using one of 4 keys, each key corresponding to a possible transformation:

- EA: to enter the first item of the file AS in memory A (this item is thus cleared from the file);
- EM: Same operation between file MVT and memory M;
- AD: to add the contents of memory M to that in memory A (clearance of memory M) and the total appears in A;
- SA: to write the contents of memory A in the file NS (clearance of memory A);

The result of each transformation appears on the screen. The subject is shown how to proceed before taking part in the actual experiment:

-item without transaction: EA - SA
-item with one transaction: EA - EM - AD - SA
-item with "n" transactions: EA -$[EM - AD]^n$ - SA
where $[...]^n$ indicates n times [...]

Finally, each subject is assigned at random to one of the two experimental devices described below.

2.2. Experimental devices

2.2.1. Device compatible with a strong semantic representation of the data

The subject, using the commands described above (EA, EM, AD, SA), only has access to the immediate states of memories A and M. After each operation EA (entry of an item) and AD (addition of a transaction), however, memory M provides him with information about the nature of the following transaction (first line of the file MVT):

(a) if the word MVT appears (ex: "6 18 MVT") it means that the item which is currently being processed (ex: Item 6 in memory A) will be involved in the next transaction. The subject must therefore type the command sequence EM - AD until further information is supplied.
(b) if, however, the word MVT does not appear, the processing of this item has been completed and the system is waiting for the next item to be processed. The subject inserts the commands sequence SA - EA and again further information is supplied (ex: "9 11 " which means here that there is no transaction for item 9).

The correct procedure can therefore be illustrated by the following rules:

IF		THEN EXECUTE
R1: presence of indication MVT	⟶	EM - AD
R2: absence of indication MVT	⟶	SA - EA

This device is compatible with a semantic representation of data in terms of transactions nested in items. The following goal stack is the result:

Goal Stack:	Procedure:
File AS updating	$[EA-[EM-AD]^{n}-SA]^{p}$
item updating	$EA-[EM-AD]^{n}-SA$
transaction processing	EM-AD

2.2.2. Device incompatible with a strong semantic representation of the data

This device is identical to the previous one except there is not indication MVT. The experiment is therefore exactly the same as before, but the item number in memory M is masked if it is different to that in memory A. The subject having just processed the transaction of one item and having entered a transaction concerning another item, without this masking, may have calculated the number of intermediary items without transactions. This would have been impossible with the compatible device. The subject could process the masking (after EA or EM) in the following way:

(a) if the transaction number in memory M is legal (ex: "6 18 6 -16") it means it is the same as the item number being currently processed (in memory A): the subject can therefore use the command sequence AD - EM until further information is supplied by the following indication.
(b) if however, the number in memory M is illegal ("6 92 XXX 50") it means a different item is being processed and that the command sequence SA - EA will lead to further information.

The correct procedure can be written by a system of rules analogous to the rules previously mentioned with just one inversion, namely of the sequence EM - AD to AD - EM:

IF		THEN EXECUTE
R1: unmasked	⟶	AD - EM
R2: masked	⟶	SA - EA

The inversion of the sequence EM - AD makes no difference to those items without transaction. They will still be processed by the sequence EA - SA. This inversion does however affect the items with transactions as these are now processed by a sequence $EA - [AD - EM]^{n} - SA$. The device is now incompatible with the previous goal stack. The subject can only identify the final transaction of an item by entering a transaction which is foreign to this item. The processing sequence of this transaction is thus interrupted: [EM - [SA -...-EA] -AD]. The nesting of the earlier goal stack no longer corresponds to a complete nesting by this procedure. At various moments the subject has to change to a higher level goal (item) whilst already seeking a lower level goal (transaction) without having accomplished it.

The subject is forced to change his semantic representation into a more successful abstract representation in which the data are structured as a list of pairs (content of memory A, content of memory M). From now on he employs simple rules of passage from one pair to another without having to consider the nature of the objects being processed (items, transactions).

The objective being to study control structures used in executing procedures, the elaboration of these procedures is deliberately accelerated by telling the subject (using either the compatible or the incompatible device) the correct procedure in terms of a semantic representation of the data.

2.3. Design

The subjects (factor S) are divided into two groups of 10 - each group corresponding to a device type (factor D). During the execution of a procedure, eight possible types of transitions (factor T) between commands are defined (Figure 2). Each

procedure being represented by a two rule system, a distinction is made between: inter-rule transitions (t1 to t4) which correspond to identifications and intra-rule transitions (t5 to t8) which are simple links between commands. The response latency is measured for each correct transition.

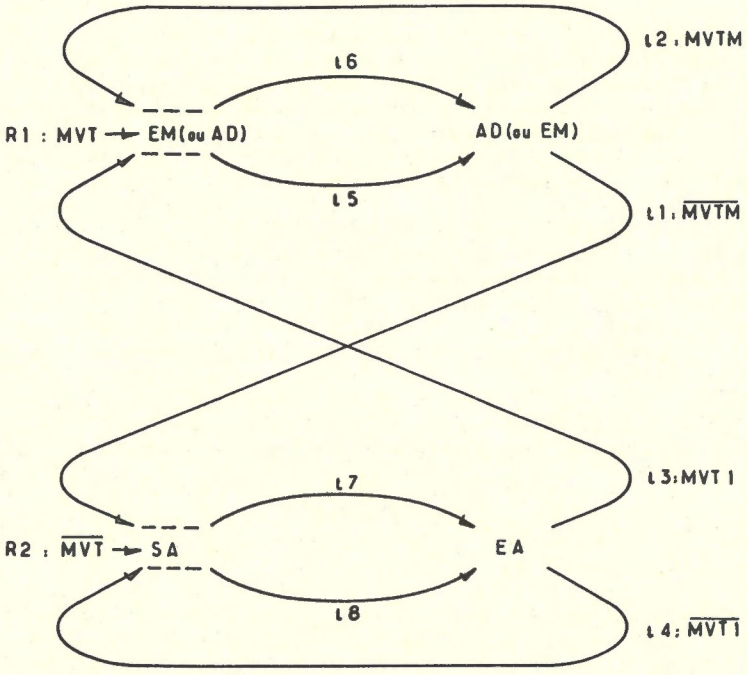


Figure 2 Transition types for the two devices. Meaning of the transitions:

t1: transition at the end of processing an item with transaction : there is no other transaction: $\overline{MVTM}$: SA —t7→ EA.

t2: transition to the following transaction: there is another transaction (MVTM) : EM (or AD) —t6→ AD (or EM).

t3: transition to processing an item with transaction: there is a first transaction (MVT1): EM (or AD) —t5→ AD (or EM).

t4: transition to processing an item without transaction: there is no transaction $\overline{MVT1}$: SA —t8→ EA

The data to be processed are divided into eight successive blocks. The frequency with which each of the four types of inter-rule transition appear is equally balanced. There are 80 transitions for each block which means that each subject executes 640 (8 x 80) transitions. The factor "Block" has been introduced to examine the effects of a potential modification of the control structure on time. The data have been analysed by analysis of variance followed by "fiducial inference" using the design formula:

$$S_{10}<D_2>*T_8*B_8$$

(subjects nested in devices and crossed with transitions which are themselves crossed with blocks). (For fiducial inference and notation of design formula, see Rouanet et al., 1976; Hoc, 1983b, c).

3. RESULTS

In order to define the control structure employed in each block and to evaluate the inter-individual variability within each group of subjects the individual protocols were analysed. As the variability proved to be very small, analysis per group for each device was completed, the results of which are presented here.

3.1. Compatible device (semantic representation)

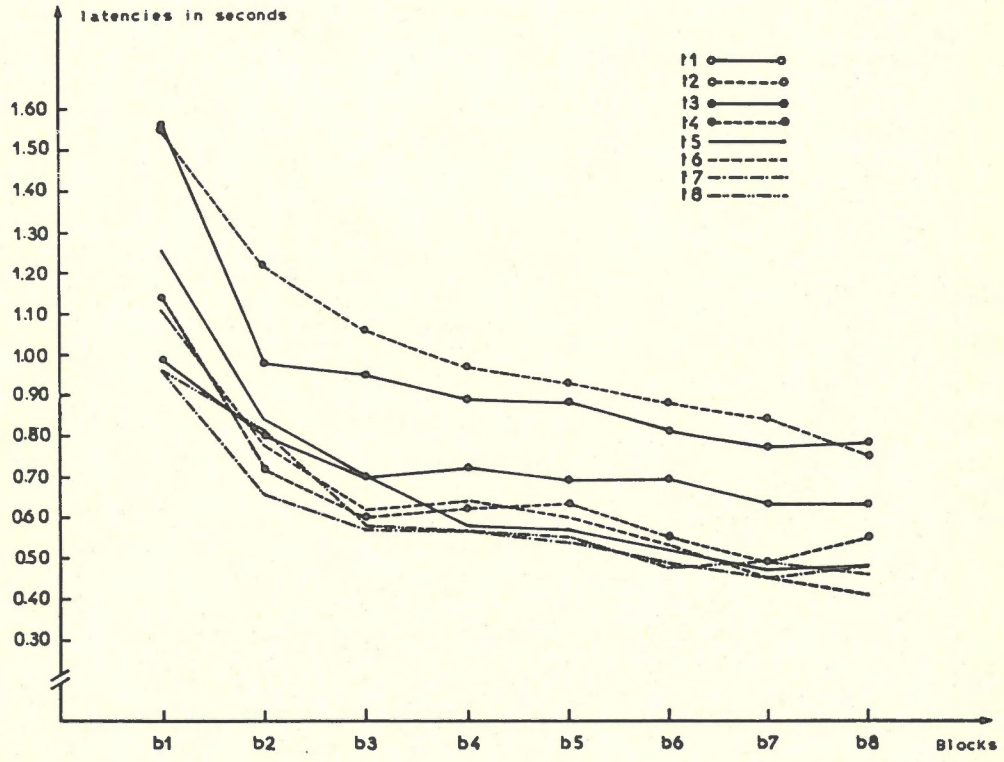


Figure 3 Response latencies for the compatible device (d1):
abscissa: successive blocks,
ordinate: mean latency in seconds.
Evolution of the latencies of the eight types of transition over all the blocks.

Figure 3 shows the evolution of the response latencies corresponding to each transition type for all the blocks. We would like to point out, that independent of the

block, all eight transition types show the same response latencies pattern. Response latencies for the intra-rule transitions (links) are the shortest and have the smallest variance. For the response latencies of the inter-rule transitions (identifications: see also figure 2):

(a) with a guarantee of .98 it is inferred that the response latencies for t3 are shorter by at least 160 msec. (22% of the latency observed for t3) to those for t4, over all the blocks.[2] Therefore, after EA the subject is prepared to process an item with transaction.
(b) with the same guarantee of .98 it is inferred that the response latencies for t2 are at least 140 msec. (21% of t2) shorter than those of t1, over all the blocks. Therefore after AD the subject is prepared to process another transaction.

In addition, we note that if the latencies corresponding to preparation phases (t2 and t3) are similar to those for the intra-rule links (t5 to t8), then the data are compatible with the hypothesis that those conditions, which correspond to the states of preparation, are not explicitly identified by the subject.

The frequency of errors is revealing in this respect: a prepared transition (t2 or t3) is more often triggered off in error (11.3% and 27.5% respectively) than an unprepared transition (t1 : 2,5%; t4 : 0%).

3.2. Incompatible device (abstract representation)

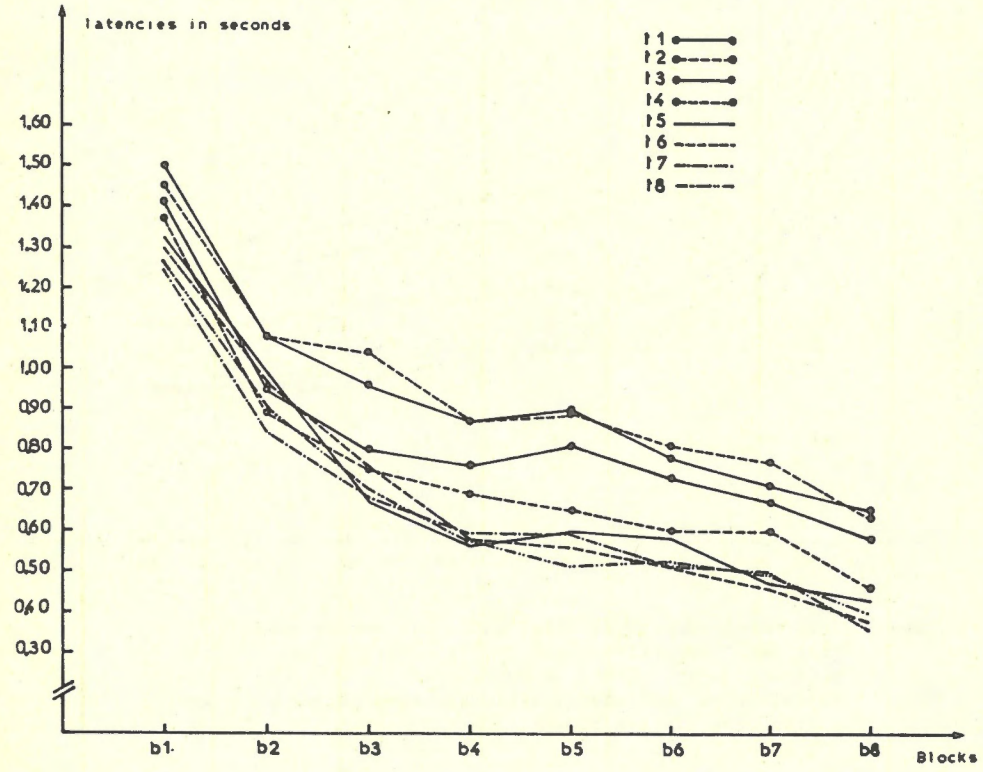


Figure 4 Response latencies for the incompatible device (d2) (see sub-title of figure 3)

Figure 4 is analogous to figure 3 for the incompatible device. The preparations observed for the compatible device have now almost entirely disappeared:

(a) response latencies for t3 and t4 are very similar to each other over all the blocks. Absolute value of the difference is smaller than 80 msec. (8% of t3 and t4) with a guarantee of .95.
(b) the same can be said for t1 and t2 for the blocks b1 to b4, and b7: the absolute value of the difference is smaller than 130 msec. (14% of t1 and t2) with a guarantee of .90. Nothing can be concluded for the other blocks which implies that abstract representation is not always stable. Error frequencies are small and similar to each other when there is no preparation.

The data are compatible with the hypothesis that the subject now explicitly identifies all the conditions.

4. DISCUSSION

The existence of different preparation states leads to a distinction between two types of identification to be found in the control structure of the procedure executed by each subject:

(a) Specific Identifications:
They are based on well defined conditions of validity and occur at precise moments in the execution.
(b) Identifications of Invalidity:
They are more diffuse and capable of halting the execution at any time to return the control to the specific identifications.

The procedures employed by the subjects can be modelised as systems of rules in which the component "condition" only brings out the specific identifications (Hoc, 1982). The model for the observed procedure with the compatible device would therefore be of the following type (transition numbers are the same as in figure 2):

t4 t8 t3 t5 t2 t6
no initial transaction → SA-EA -EM -AD -[EM - AD]^n
no following transaction → SA-EA -EM -AD -[EM - AD]^n
t1 t7 t3 t5 t2 t6

We note that t2 and t3, which correspond to the two preparation states, are processed as simple links which are only interrupted when invalidity (considered a "demon") is identified. As these identifications of invalidity are diffuse, it may happen that they don't operate correctly and so don't prevent the release of inappropriate actions. This results in mistakes such as these examined above.

This procedure does not include conditionals, such as those known in informatics: all the identifications do not by way of specific identifications (left side of the rules) such as was the case for the incompatible device.

What now remains to be considered is the difference in the control structure between a situation in which the semantic representation of the data can be retained and one in which the subject has to adopt a more abstract representation.

In the first case (semantic representation), there is double preparation: for an item with transaction and for the presence of another transaction. Two reasons can be put forward to explain this preparation for an item with a possibility of transaction:

(a) a phenomenon of typicality - this property is more typical in an updating task
(b) a more satisfactory nesting of the transactions in the items. (items without transaction are not considered).

On the other hand the preparation for another transaction concerns the absence of specific identifications of the condition for continuing the iteration (that of processing the transactions).

It should be noted that the semantic features of the data to be processed (items, transactions) create effects which are extremely resistant to any temporal adaptation. This could have lead for example to the removal of the preparation states for the later blocks.

In the second case (abstract representation), it is precisely this temporal adaptation which works with the incompatible device. As a result of adopting an abstract representation of the data in the form of a list of pairs, with rules for passing from one pair to another, the phenomena observed in the previous situation disappear:

(a) the lack of typicality and of a nesting form of representation take away the necessity for the preparation of a particular item.
(b) this same lack of nesting removes the iteration on the transaction and, by that, the absence of any explicit identification of a stopping condition.

5. CONCLUSION

Although this experiment has only a limited significance, we feel that it highlights the need for a deeper study of the control structures used in carrying out procedures so that the mechanisms of becoming aware can be better understood. The construction of a computer program is never done from a tabula rasa - the subject always knows some parts of the procedure and, using the mechanisms of becoming aware, he will try to express them in the programming language.

This experiment certainly shows that the equivalent of computer tests can be found in the control structure of a procedure carried out by a subject. But at the price of an abstraction which it is not always possible to achieve. Programming problems convey semantic representations which are not easily made abstract. These phenomena can be used to understand beginners' difficulties, the effects of typicality are, for example, the source of certain programming errors even for experienced subjects, e.g. authors of programming manuals, as it has been shown by Lesuisse (1983).

It has also been noted that even when a procedure seems algorithmic, the subject will work according to a heuristic which ensures a certain economy of identification by treating the procedure sequentially. This can be seen in some of the results obtained by Miller (1981) in his studies on programming in natural language.

The question to be asked is whether it is fruitful to construct programming languages which use "natural" control structures. On one hand these "natural" structures are not always reliable but on the other it is probably impossible to be aware of certain components of these structures. If the presence of diffuse identifications of invalidity is a viable hypothesis, they are doubtless outside our realm of awareness.

In order to answer these questions and to improve our definition of the basic constructions of our own control structures, further research is necessary. The tools introduced by artificial intelligence could be very useful in this type of analysis.

FOOTNOTES

1) This research was supported by the "Agence de l'Informatique"

2) The meaning of this formulation is the following: Without any other information than the one provided by the two samples, an uncertainty distribution on the set of possible values for the difference between the two population means can be defined. From this distribution, the following statement (fiducial conclusion) can be derived: "the probability that the population mean latency for t3 is shorter by at least 160 msec. to the population mean latency for t4 is .98": $p^*(\mu_4^* - \mu_3^* > 160\text{msec.}) = .98$. To evaluate its importance the difference (160 msec) is related here to the latency observed for t3: 22%.

REFERENCES

Cordier, F.,Dubois, D. (1981). Typicalitè et reprèsentation cognitive. Cahiers de Psychologie Cognitive, 1, 299-333.

Green, T.R.G. (1980). Ifs and thens: is nesting just for the birds? Software Practice and Experience, n. 10.

Green, T.R.G.,Payne, S.J., Van der Veer, G.C. (1983). The psychology of computer use. Academic Press, London.

Hoc J.M. (1982) Reprèsentation des donnèes et structure de contrôle d'un processus de traitement. Cahiers de Psychologie Cognitive, 2, 389-419.

Hoc J.M. (1983a). Analysis of beginner's problem-solving strategies in programming. In Green et al.,

Hoc J.M. (1983b). Evaluation of different modalities of verbalisation in a sorting task. International Journal of Man-Machine Studies, 18, 293-306.

Hoc J.M. (1983c). L'analyse planifièe des donnèes en psychologie. PUF, Paris.

Lesuisse, R. (1983). Analyse des raisonnements faits et des erreurs commises dans des programmes publiès de recherche dichotomique. Le Travail Humain, 46, 239-254.

Miller L.A. (1981). Natural language programming: styles, strategies, and contrasts. Perspectives in Computing, 1, 22-33.

Piaget, J. (1974a). La prise de conscience. PUF, Paris.

Piaget, J. (1974b). Rèussir et comprendre, PUF, Paris.

Richard, J.F. (1980). L'attention, PUF, Paris.

Rouanet, H., Lèpine, D., Pelnard-Considère, J. (1976). Bayes-fiducial procedures as practical substitutes for misplaced significance testing: an application to educational data. Advances in psychological and educational measurements. D.N.M. de Gruiter, L.J.T. van der Kamp, H.F. Crombag (eds). Wiley, New York.

Sime M.E., Arblaster, A.T., Green, T.R.G. (1977). Reducing programming errors in nested conditionals by prescribing a writing procedure. International Journal of Man-Machine Studies, 9, 119-126.

Van der Veer G.C., Van de Wolde, G.J.E. (1983). Individual differences and aspects of control flow notations. In Green et al.

COGNITIVE ERGONOMIC RESEARCH AT SAPU, SHEFFIELD

T.R.G. Green
MRC/ESRC Social and Applied Psychology Unit [1]
University of Sheffield
Sheffield S10 2TN
U.K.

This is a report on recent research by myself and my colleagues, Max Sime, Stephen Payne and David Gilmore. Where I say "we" and "our" it refers to all of us. I hope I have not misinterpreted their ideas too much.

Previous research at this Unit into the causes of difficulty in comprehending programs led us to the conclusion that it is useful to regard programs in the same light as other forms of presentation of complex information, and to ask how easy is it to extract necessary information from them. This view point emphasises the role of structure: the program structure must be easily perceived, and it must make it easy to perform the user's task given the usual human abilities and disabilities. Structure must be well-specified, visible, and appropriate.

In the first section of this paper I shall briefly outline the course of our work on program comprehension, in order to establish our views on structure. The following sections describe recent research at this Unit into the causes of difficulty in learning and using text editors. We believe that the notations of command languages and of programming languages need to satisfy very similar requirements as regards visible and appropriate structure. The final section offers some conclusions, necessarily tentative.

1. COMPREHENDING SMALL PROGRAMS: POOR NOTATION CAUSES PROBLEMS

The early programming work at this Unit is often linked with the use of the 'Hungry Hare', a simple card-sorter with some lights on it; this purpose-built laboratory device has aroused both admiration (du Boulay and O'Shea and Monk, 1981) and disdain (Sheil, 1981). Early experiments established that students with no programming experience found it easier to write simple conditional programs using a particular form of nested syntax than using an unconstrained GOTO language (Sime, Green and Guest, 1973, 1977); in particular, with one syntax (called Nest-INE) they cleared up bugs more easily. Subsequent experiments attempted to establish an explanation of these results in a form that was sufficiently general and powerful to extend to other constructions in programming. Unfortunately these experiments are often regarded simply as comparisons of conditional designs, rather than comparisons of classes of information structure in which the conditional serves as a representative of many similar structures.

There are strong indications that the effects are caused by differences in the ease of extracting information from programs, and in particular the ease of extracting 'circumstantial' information: that is, discovering the circumstances which cause a conditional program to behave in a specified way. Among these clues, Green (1977) found that when professional programmers answered circumstantial questions about programs, the different syntactic designs greatly affected response times; whereas when they answered sequential questions (the inverse of circumstantial - that is, given the circumstances they found the action) the different syntaxes made little difference; a similar result, obtained with a quite different paradigm, was described by Green (1980). Another important indication was that when we returned

[1] Henceforth, MRC Applied Psychology Unit, 15 Chaucer Road, Cambridge.

to asking novices to write programs for the Hungry Hare, using the worst of our syntax designs, the GOTO or Jump language, we found that their programs were more likely to be correct first time if they were constrained to use GOTOs to simulate nested conditionals instead of using them haphazardly - BUT they were no better at correcting their mistakes than the unconstrained group (see Arblaster et al., 1980, for summary),

The obvious explanation is the one we have proposed: conventional, procedural, programming language designs favour the extraction of sequential information, but circumstantial information can be made more available by using a well-structured design to make the information-gathering operations simpler, and making the information more visible with cues (here provided by Nest-INE). Can other explanations be found? Our experiments clearly showed that 'good structure' on its own was insufficient, and indeed studies by Van der Veer and Van de Wolde (1983) and unpublished studies by ourselves have found cases where 'good structure' is counter-productive. In his review paper, Sheil (1981) suggests that variations in program length, caused by the different syntax designs, may have caused these effects, but since the same programs were used for both types of question in the studies described by Green (1977) and Green (1980) I am at a loss to understand his reasoning. It is equally hard to understand how his explanation would address the results on correction of errors using constrained and unconstrained GOTOs.

Pursuing our explanation leads to the following position. (1) The activities of programming clearly involve both reasoning about programs, and discovering the facts upon which reasoning is based. (2) Discovering the facts frequently means extracting information from written programs (or other notations, such as specifications, manuals, post-mortem dumps, etc.). (3) Different classes of information structure highlight different types of information; in particular, procedural programming languages highlight sequential information, and declarative ones highlight circumstantial information. (4) The availability of information is therefore limited by the difficulty of extracting it from the given structure, and when there is a mismatch - e.g. when requiring circumstantial information from a procedural program - performance will be worse; this will create errors, slow up performance, impair reasoning, encourage programmers to guess rather than make certain, etc. (5) The problems can be ameliorated by adding cues; what the Nest-INE syntax did was to add cues to help in the extracting of circumstantial information. (6) Simple models of information extraction, using such mental operations as searching for labels, negating predicates, performing parsing manipulations, etc., can not only account for the results but can also extend them to many other familiar aspects of programming language design, such as parameter passing.

1.1 EXPERIMENT I: CUES AND STRUCTURE

In an experiment by Gilmore and Green (1984), we have gone some way to justify some of these grand claims by a direct test of the 'mismatch hypothesis'. A short algorithm was coded in four ways: either procedural or declarative and either with cues or without cues. Subjects (non-programmers, to avoid effects of prior experience) were asked questions that required either sequential information or circumstantial information. Some of the questions were answered with the program in front of them, some were answered from memory of the program. Full details of the study cannot be given here, but Table 1 shows a sample of the results, taken from the recall stage. The figures show quite clearly that cues improved performance in mismatch conditions, as predicted. Moreover the differences between uncued match and mismatch conditions were in complete agreement with our predictions, although in the cued conditions our predictions were slightly upset.

Although our explanation is not perfect, it is intuitively 'obvious' and it successfully predicts effects unknown to any competing theory. (Examples of competing theories of program comprehensibility that cannot explain these results include the 'syntax-semantics' model of Shneiderman and Mayer, 1979; the structured programming

school; and the software science school founded by Halstead, 1977). It would be extremely interesting to extend this line of research to much larger programs, to see how it affects the work of professional programmers, but the practical difficulties have prevented that. However, the important point is that these effects are, if our theories are correct, to be associated with classes of information structure and with the specific tasks to which they are put, rather than with choosing language F or language P.

TABLE 1
Percentage Correct, Experiment I, Stage 2:
Answering Questions from Memory

Mismatch conditions:	No cues	Cues	Effect of cues	Predicted effect
Procedural	64	78	14	+
Declarative	60	73	13	+
Match conditions:				
Procedural	68	64	-4	o
Declarative	72	74	2	o

2. WHAT MAKES COMMAND SYNTAX HARD?

Command languages usually have opaque syntax. Moreover, they are usually critically dependent upon details of punctuation or spacing, for reasons which are not obvious to novices. Some brief examples:

```
s/a\/&c/p\/q\&r&/
```
How the Unix 'ed' editor expresses "change 'a/?&c' to 'p/q&c plus whatever the original string was", where '?' is a wildcard symbol

```
PIP A:TG*.* B:[VZ]
```
CP/M system instruction "copy all files on drive A matching the wildcard name TG*.* onto drive B, masking off bit 8 and verifying the copy"

```
ATTACH(OLDPL,$:APPLSRCES.CERNPROGLIB(*MT),ST=S19)
```
This is a magic Job Control Language command taken at random from a computer centre manual. Novices often have to cope with a good deal of this sort of thing, usually with little or no understanding of the structure of the command - not even at the level of where spaces would be allowed, if anywhere.

There are many examples to be found. In modern systems great effort has been put into avoiding the need for these horrors; mainframe operating systems may still need to be addressed in such language, but in the micro world display editors have replaced context editors and menus have superseded command strings. Nevertheless command language syntax still has its uses. Because it is terse it is particularly appropriate for use over slow telecommunications systems (try using a display editor at 300 baud!), for cases where information must be densely packed, for the description of batch jobs such as stream editing tasks, and for cases where menus are too slow or use too much system memory. It also makes less demand on system programmers, and in consequence systems built in a hurry are likely to interact through command languages. For all these reasons, command language systems are likely to remain alive and well for many years.

Ledgard et al. (1980) have demonstrated impressive improvements in the usability of an editor by rewriting commands into what they called an 'English-like' form:

```
FIND:/TOOTH/;-1      =>     BACKWARD TO "TOOTH" or BA "TOOTH"
RS:/KO/,/OK/;*       =>     CHANGE    ALL "KO" TO "OK" or CA "KO" T "OK"
```

One component of the improvement may indeed be English-likeness, but clearly the 'improved' notations do more than resemble English; they also display much more clearly the underlying syntactic structure. Instead of the homogeneous strings of inscrutable symbols, Ledgard et al. have divided their commands into four clearly distinct fields, and have used quotation marks to show that two fields are special ones to be taken literally. Since their users in fact used the abbreviated versions the resemblance to English is much less striking than the improved perceptual clarity.

2.1 EXPERIMENT II: PERCEPTUAL PARSING

Our first experiment (Payne, Sime and Green, 1984) investigated the improvement to be gained from what we called 'perceptual parsing' - that is, providing cues, as Ledgard et al., did, to display the underlying syntax.

We argued that evidence from many sources shows that typographical cues can help the comprehension of complex textual material, including instructional text (Hartley, 1978), tables (Wright, 1977), diagrammatic instructions (Szlichcinski, 1979) and sheet music (Sloboda, 1981). These cues can be interpreted as methods for mapping the structure of the information onto the spatial layout or other peceptual cues (Green and Payne, 1982). Forceful arguments have been presented by Bever (1970) to suggest that even natural language is parsed more easily when perceptual strategies can be used to supplement or to replace genuine syntax parsing. There is also a certain amount of evidence that conventional programming languages are understood more easily when the structure of programs is cued at the perceptual level (Sheppard et al., 1981; Norcio, 1982; Miara et al., 1983).

The evidence at present therefore indicates that perceptual structure cueing does indeed improve performance; but at present the evidence comes only from extremely complex situations. Our experiment was designed to discover whether the effect also extended to extremely simple situations. If so, one may be confident that it extends even to simple commands in simple text editors.

A very simple editing language was devised for editing single line phrases, using paper and pencil commands. A typical command was

```
3dBc
```

meaning "find the 3rd 'd' and put a 'c' before it". Two dialects were used, in one of which the operation codes were always in upper case, as in the example. In the lower case dialect the same command would read

```
3dbc
```

Five operations were possible, as follows:

insert after	a or A
insert before	b or B
change	c or C
delete	d or D
swap	s or S

Examples:

1vau	find the first v and insert u after it
3zbh	find the third z and insert h before it
2wcy	find the second w and change it to y

3xd	delete the third x
2ms	swap (exchange) the second m with the following letter

Ten subjects (students) learnt each dialect. Three types of task were devised, and for each task, each subject completed six sets of eight problems, the order of sets being controlled across subjects and tasks. The upper case dialect will be used for illustrations.

Task 1, command generation: Subjects had to write down a command to effect an alteration marked in a short phrase, e.g.

PROBLEM:	the paper ppresents
SOLUTION:	3pD

Task 2, command decoding: Given a seven letter string together with a command, subjects had to mark the effect of the command, e.g.

PROBLEM:	kryhcb	2rBc
SOLUTION:	krychcb	

Task 3, inverse commands: Subjects were shown a command, and a string that had resulted from applying that command to an unseen string. They had to generate a new command that would undo the effect of the given command and return the original string, e.g.

PROBLEM:	2dAs	hdssdsw
SOLUTION:	3sD	

We investigated two hypotheses. First, that the upper case commands would be easier because the command strings could more easily be parsed into literal fields and command field; second, that extra difficulty would be caused when the letters in the literal field were the same as letters occurring in the command field, e.g.

2aAf

It even be the case that extra difficulty would be caused when letters in the literal field were ones that might potentially occur in the command field, e.g.

2aCb

We shall refer to these two possibilities as 'perceptual parsing' and 'command-literal overlap'. It is difficult to separate them completely, but the six sets of problems presented different degrees of overlap to the subjects. (See Payne et al., 1984, for a fuller description of the design.)

Both times and errors were recorded. The times for the subjects using the upper case dialect were slightly faster, but the difference was not statistically significant. In each of the three tasks, however, the upper case group made far fewer errors than the lower case group (Table 2), and these differences were statistically highly significant. There was no sign of any differences between the various amounts of overlap.

The differences were statistically highly significant for each task, using both parametric and distribution-free tests (see Payne et al., 1984, for details). We conclude from this experiment that perceptual structure cueing makes a real difference to performance even in a very simple command language.

The simplicity of the language is most important. Nobody would be surprised if perceptual cues made it easier to use extremely complicated systems. What we have shown is that perceptual effects persist into regions of such simplicity that one would surely not expect them, regions where system designers might easily say "Surely nobody could get that wrong". The implication is that thought should

ALWAYS be given to providing perceptual cues, wherever the command language contains more than a shred of syntactic structure.

TABLE 2
Error Scores, Experiment II

Overlap conditions:		1	2	3	4	5	6
Task 1	lower case	9	7	9	16	10	9
	upper case	0	4	4	5	2	4
Task 2	lower case	3	1	1	3	3	3
	upper case	0	0	0	0	0	0
Task 3	lower case	10	7	8	9	10	6
	upper case	3	1	0	2	1	3

2.2 EXPERIMENT III: A SPEECH-DRIVEN TEXT EDITOR

If we extend the argument, it leads to the suggestion that syntactically distinct fields of commands should be made as perceptually distinct as is conveniently possible. An attractive possibility is to combine our interest in command language design with current interest in speech recognition devices and what they might be used for. Our next study therefore used a simple device, capable of recognising a small number of spoken words with tolerable accuracy, for controlling a text editor. The device in question was a Heuristics Speechlink attached to an Apple computer, and obtaining usable recognition performance from it was a study in itself which we shall not describe here (see Green, Payne, Morrison and Shaw, 1982).

For this experiment, performed by Morrison, Green, Shaw and Payne (1984), we built an editor that could be driven either by spoken commands or by commands entered with function keys; in either case, the associated literal strings were typed in the conventional manner. We hypothesized that speech-driven editing would be an improvement over pure keyboard editing, since it would eliminate confusion between command and literal and thereby lower the mental workload.

We also compared two language designs. Both languages were based around context editing (e.g. "find the string APPLE"), but in one design, called the Many Commands editor, the language included a large number of different commands, each of which was relatively weak; in the other design, the Few Commands editor, there were fewer commands to choose from, but each one was more powerful. The difference was most apparent with the cut-and-paste operation. Suppose we wish to move the block of lines starting with the string APPLE and ending with the string PEAR, placing them in front of the line containing the string ORANGE. In the Many Commands editor, at least 6 commands would be needed:

locate the cursor at the start of block:	LOCATE APPLE
place Start-Marker	BEGINNING
find PEAR:	LOCATE PEAR
place Finish-Marker:	FINISH
find ORANGE:	LOCATE ORANGE
move block:	TRANSFER

In the Few-Commands editor, however, only one command would be needed:

TRANSFER APPLE PEAR ORANGE

Remember that all the commands, LOCATE etc., were either spoken or else were single function keys, while the parameter strings APPLE etc., were typed in full.

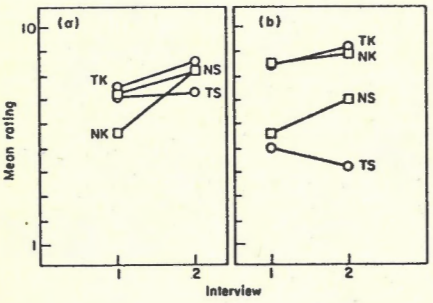
We suggest that the Many Commands (MC) editor will be easier because the user receives knowledge of results at each step. In particular, we believed that the text editor would be particularly difficult with the Few Commands (FC) editor because the potential confusion between literals and commands would interact with the increased mental load.

We compared the performance of four groups of subjects: professional typists and non-typists using the MC and FC editors. Each subject attended four sessions lasting approximately 1 hour each, during which she edited documents presented to her on-screen and made changes as indicated on a marked-up printed copy of the document. All the subjects used their editor, either the MC or FC version, both as a speech-driven tool and also as a keyboard-only tool. In the speech mode, subjects spoke the commands into a microphone, first depressing a foot switch; then the parameter strings were typed on the keyboard. (If the computer failed to recognise the command it displayed the word PARDON?) In the keyboard mode the subject pressed a function key instead of using the microphone. Parameter strings on the keyboard were ended in both conditions by pressing the RETURN key. Accuracy, speed, and - most important - subjective ratings were all recorded, the ratings being obtained from questionnaires and from structured interviews at the end of the first and fourth sessions.

The results of this study were quite mixed. Times differed little; speech input tended to be slower, but that might well have been because the speech recognition device was slow to operate. There was no significant advantage for either the MC or the FC editor. Not surprisingly, speeds increased with practice, and the typists were 'almost significantly' faster than the non-typists.

Error frequencies were also quite similar for the two editors, but there were strong interaction effects. With the MC editor non-typists managed about equally well with speech or with keyboard, but typists made significantly fewer errors when using speech. With the FC editor a different interaction showed - non-typists using speech made fewer errors than typists using speech. The suitability of speech appears to depend upon who is using it for what purpose.

FIGURE 1



Overall ratings of attitudes averaged over subjects. Interview 1 took place after one hour of practice, interview 2 after four hours. Highest ratings express most satisfaction.

(a) = MC editor, (b) = FC editor, TK = typists + keyboard, TS = typists + speech, NK = non-typists + keyboard, NS = non-typists + speech.

Subjective assessments strongly reinforced this conclusion. In particular, typists using the FC editor grew to dislike it quite firmly (see Figure 1). Even non-typists grew to prefer keyboard input! Comments on the problems of switching between modes, speech to typescript, were frequent.

It would seem, therefore, that our arguments concerning perceptual parsing do not extend as far as this. When extra effort (pressing a foot switch, changing mode between speech and typing) is required, plus extra time for decoding the spoken command, subjects prefer to stick to the keyboard. However, the experiment confirmed

one of our hypotheses, namely that the FC editor would create additional difficulties for users and would exacerbate any existing problems.

3. WHY DO USERS MAKE MISTAKES?

Everyday observation suggests that some computer systems entice their users into mistakes more often than others: try counting how often you hear people exclaim "Dear me, I'm ALWAYS doing that!!". The most interesting problems are 'slips' - errors made by experts, sometimes defined as 'actions not as planned' (Reason, 1979) or as 'the error that occurs when a person does an action that is not intended' (Norman, 1981). There have been recent attempts to classify and explain persistent slips, which would be especially fruitful if they led us to design 'non-slip' systems. Since slips are, by definition, made by experienced users, the available theories concentrate on what happens inside users as they become expert.

Reason (1979) considered the detailed consequences of the shift from 'controlled' or 'closed-loop' control, as in novices, to 'automatic' or 'open-loop' control, as in experts. Diaries of slips in everyday life, kept by volunteers, together with theoretical considerations of the nature of the motor program theory of skilled performances, led him to classify four types: *test failures*, which occur when the open-loop mode of control coincides with a decision point in a motor program where the strengths of competing motor programs are markedly different; *discrimination failures*, when, during open loop control, the wrong stimulus is accepted (e.g. salt for sugar); *forgetting previous actions*, possibly caused by switching from open-loop control to closed during a critical phase of an action sequence; and finally *omissions*, tending to occur when unexpected events took the place of expected events.

A somewhat more detailed analysis, applying especially to typing errors, was offered by Norman (1981, 1983). In his ATS (activation-trigger-schema) model, a skilled sequence starts with a parent schema, representing the initial goal, which can invoke child schemas as required to achieve subgoals. Each schema is provided with an activation level and with a set of specific conditions that are required for it to be triggered, although an exact match is not required. At any one time during the execution of a high level skill, many schemas will be active, competing with each other. The ATS analysis is most at home in explaining *capture errors*, which resemble Reason's discrimination failures, since they occur "when there is overlap in the sequence required for the performance of two different actions, especially when one is done considerably more frequently than the other". Norman offers a number of examples of slips in interacting with computers. For instance, in a well-known text editor "the command ':w' means to write the file. ':q' quits the editor ... and the combined sequence ':wq' writes, then quits". The ':wq' soon gets automatised and captures the ':w'. A solution, according to Norman, would be to use a completely different sequence, such as ':zz;, in place of ':wq'.

3.1 DISCRIMINATION FAILURES AND THE USER'S MODEL OF THE TASK

Research at Sheffield has attempted to extend these models of slips (Green et al., 1984). Two principal extensions have been made. First, it is clear that the user's representation of the command language and the editing task will affect the number, frequency, and type of slips. We have argued elsewhere (Payne and Green, 1983; Green and Payne, 1984) that the user's knowledge of the command language includes knowledge of family resemblances between rules, allowing users to work out one rule from their knowledge of other rules - as long as the language is consistent. Payne (1984) has extended this line of argument, developing a generative model of the user's knowledge of the task structure in which features of the task are used to determine the action of the generative grammar. For instance, one part of the task structure may be modelled as a production rule such as this:

```
IF a goal is to correct a word
AND the word is to the right of the cursor on the same line
THEN move [forward, word]
```

The square brackets mark features, used subsequently to direct the generation of the required command. In the Mince editor, that might be CTRL-W, derived as follows:

```
move[forward, word] -> symbol [forward] + letter [word]
symbol [forward] -> CTRL      (other rules specify symbol [backward], etc)
letter [word] -> W            (other rules specify letter [character], etc) .
```

We postulate, admittedly with no evidence at present, that discrimination failures may occur during the derivation of task actions. In particular, the interpretation stage during which the task 'move [forward, word]' generates the required action may be subject to slips involving features. If so, typical slips would be to move in the wrong direction or by the wrong size unit, in the system we have mentioned; but in other systems, where the structure of the task-action relationship is different, a different feature set would have been learnt, different derivations would be used, and different slips would occur.

This model, we observe, makes different predictions from Norman's in some respects. In particular, in the example of Norman's cited above our model would attach the feature [finished] to certain tasks, such as quitting the editor and (probably) to writing the edited document to file. That feature would still be attached to Norman's proposed new command, ':zz', and therefore according to our model slips would still occur, although at a reduced level compared to the version in which the action sequences overlapped.

3.2 INTERNAL STRUCTURE OF ACTION SEQUENCES

It is inconceivable that chains of actions are formed like rows of beads, with no internal structure. Numerous studies have demonstrated that patterning is swiftly perceived in a variety of contexts, and many attempts have been made to characterise the underlying psychological processes. For instance, Restle (1970) developed a theory of hierarchical representation. Given an alphabet 1-6 and the basic sequence χ = (1, 2), the operation R ('repeat of χ') produces the sequence 1 2 1 2, the operation M ('mirror') produces the sequence 1 2 6 5, and the operation T ('transposition') produces 1 2 2 3. Then M(T(R(T(1)))) describes the sequence 1 2 1 2 2 3 2 3 6 5 6 5 5 4 5 4. There has been considerable activity in this area.

Norman's ATS typing model contains provision for 'repetition' and 'alternation' operators, but it is evident that much more advanced patterns are perceived. No doubt designers of command languages would like their users not to pay attention to such distractions as the internal patterning of response sequences. Unfortunately that is not easy, and as a skill becomes automatised it is likely that patterning will make itself felt by distorting the user's representation of the language.

An example of such distortion occurs, we believe, with the CP/M editor called VEDIT, distantly related to TECO. In command mode, typical commands include the following:

```
V       B       F...@       S...@...@
```

These respectively switch mode, find the beginning of the text, find a specified string terminated by the @ symbol, and substitute one string for another - each of the strings being terminated by the @ symbol. Commands can be strung together thus:

```
BFwinter@Ssnow@sunshine@@
```

meaning "go to the beginning of the text, look for the string 'winter', and change to next 'sunshine' to 'snow'".

In VEDIT each command string ends with two occurrences of the @ symbol. Needless to say this fact rapidly gets automatised, since direct repetition of symbols is a powerful component of the internal structuring of sequences, and creates remarkable difficulties in learning to construct complex command chains. The problem is made worse because the grammar is somewhat inconsistent; if the last command of a chain is V then the user must hit @ twice, but if the last command is F or S then that command already incorporates one @ (to terminate the parameter string) and so only one extra @ is required. It is doubtful whether the designers made much attempt to formalise the command language of VEDIT!

A simple solution might be to use an entirely different symbol to terminate the command string. More generally, designers of command language systems should be aware that users build internal structures of their more common action sequences. When the internal structure is likely to distort their view of the language they will run into difficulties, either by becoming confused about how the language works or else by making large numbers of slips.

4. TEXT EDITOR DESIGN

It is a well-established fact of cognitive psychology that we humans impose structure upon the world. We do not treat each thing as unique and alone, but instead we search for or we invent groups and families, resemblances and subsets, patterning and criss-crossing resemblances between one thing and another. The 'take-home' message of the recent experiments at SAPU Sheffield, is that structure affects performance in human-computer interaction, as it does everywhere else. Even in very simple tasks we are helped by perceptual cues to structure. Even in very well-learnt tasks (or perhaps especially in those) we are disturbed and we make mistakes if the system entices us into inappropriate structures. In their remarkable and pioneering book, Card, Moran and Newell (1983) give structural effects much less prominence than they deserve.

Another problem area in text editor design is the input system. Our work with a speech recognition device showed us that it was not immediately liked, and that it was liked still less after 4 hours' use. Apparently subjects disliked switching between speech and type. This immediately reminds us of the fashion for mice as input methods. The fashion seems to have been started by experiments with a small number of subjects performing rather specialised task, reported by Card et al. (1983) and elsewhere; although publicity managers may adore the mouse, and although they may be excellent for freehand drawing, as pointing devices within typescript they suffer because hands have to leave the keyboard. It would be extremely interesting to see a proper evaluation of the mouse.

In fact the current fascination with mice, with speech recognition, and with high-resolution display technology is hiding some deeper problems in text editor design. Here are two to end on.

(1) The low-level functionality of editors is often poor. They cannot interact gracefully with the operating system, for instance; and typically their operations are specified in terms that are alien to users, such as character sequences instead of word sequences, which makes many operations unnecessarily difficult (Green, in press, analyses four editors in detail from this viewpoint).

(2) The detailed behaviour of many editors is illogical and inconsistent, and therefore is hard to describe accurately and succinctly. Consequently users get nasty surprises, either from their imperfect understanding of vague descriptions or else from bugs left lurking in vague code. Sufrin (1982) describes how an editor can be implemented from a rigorous formal specification and can be proved correct; this promising development shows that the structure of an editor can now be well-specified. If, in addition, we can create rules that will produce structures that are also appropriate to human use, and, where necessary, highly visible, we shall indeed have made progress.

REFERENCES

Arblaster, A.T., Sime, M.E. and Green, T.R.G. (1980). Jumping to some purpose. Computer Journal, 22, 105-109.

Bever, T.G. (1970). The cognitive basis for linguistic structures. In Hayes, J. (ed.) Cognition and the Development of Language. New York: Wiley.

Card, S., Moran, T.P. and Newell, A. (1983). The Psychology of Human-Computer Interaction. Erlbaum.

Du Boulay, B., O'Shea, T. and Monk, J. (1981). The black box inside the glass box: Presenting computing concepts to novices. International Journal of Man-Machine Studies, 14, 3, 237-250.

Gilmore, D.J. and Green, T.R.G. (1984). Comprehension and recall of miniature programs. International Journal of Man-Machine Studies, 20, in press.

Green, T.R.G. (1977). Conditional program statements and their comprehensibility to professional programmers. Journal of Occupational Psychology 4, (50), 93-109.

Green, T.R.G. (1980). IFs and THENs: Is nesting just for the birds? Software Practice and Experience, 10, 373-381.

Green, T.R.G. (in press). Global search and replace facilities: A detailed study of four CP/M text editors. Behaviour Information and Technology, in press.

Green, T.R.G. and Payne, S.J. (1982). The woolly jumper: Typographic problems of concurrency in information display. Visible Language, 16, 391-403.

Green, T.R.G. and Payne, S.J. (1983). Organization and learnability in computer languages. International Journal of Man-Machine Studies, 20, in press.

Green, T.R.G., Payne, S.J., Morrison, D.L. and Shaw, A.C. (1982). Friendly interfacing to simple speech recognizers. Behaviour Information and Technology, 2, 23-38.

Green, T.R.G., Payne, S.J., Gilmore, D.J. and Mepham, M. (1984). Predicting expert slips. Proceedings of INTERACT '84, 1st IFIP Conference on Computer Human Factors.

Halstead, M.E. (1977). Elements of Software Science. New York: Elsevier.

Hartley, J. (1978). Designing Instructional Text. London: Kegan Paul.

Ledgard, H., Whiteside, J.A., Navarro, J.A. and Shneiderman, B. (1983). Comprehensibility of programs as a function of identation. Communications of the ACM, 23,556-563.

Morrison, D.L., Green, T.R.G., Shaw, A.C. and Payne, S.J. (1984). Speech-controlled text-editing: Effects of input modality and of command structure. International Journal of Man-Machine Studies, 20, in press.

Norcio, A.F. (1982). Identation, documentation and programmer comprehension. Proceedings of ACM Conference "Human Factors in Computer Systems", Gaithersburg, Maryland.

Norman, D.A. (1981). Categorization of action slips. Psychological Review, 88, 1-5.

Norman, D.A. (1983). Design rules based on analyses of human error. Communications of the ACM, 26, 254-258.

Payne, S.J. (1984). Task-action grammars. Proceedings of INTERACT '84, 1st IFIP Conference on Computer-Human Factors.

Payne, S.J. and Green, T.R.G. (1983). The user's perception of the interaction language: A two-level model. Proceedings CHI '83 ACM/IEEE Conference on Human Factors in Computer Systems.

Payne, S.J., Sime, M.E. and Green, T.R.G. (1984). Perceptual cueing in a simple command language. International Journal of Man-Machine Studies, 20, in press.

Reason, J. (1979). Actions not as planned: The price of automatization. In G. Underwood (ed.), Aspects of Consciousness, Vol I, London: Academic Press.

Restle, F. (1970). Theory of serial pattern learning: Structural trees. Psychological Review, 77, 481-495.

Sheil, B. (1981). The psychological study of programming. Computer Services, 13, 101-120.

Sheppard, S.B., Krues, E. and Curtis, B. (1981). The effects of symbology and spatial arrangement on the comprehension of software specifications. Proceedings 5th International Conference on Software Engineering, 207-214.

Shneiderman, B. and Mayer, R.E. (1979). Syntactic/Semantic interactions in programmer behaviour: A model and experimental results. International Journal of Computer and Information Sciences, 8, 219-238.

Sime, M.E., Green, T.R.G. and Guest, D.J. (1973). Psychological evaluation of two conditional constructions used in computer languages. International Journal of Man-Machine Studies, 5, 105-113.

Sime, M.E., Green, T.R.G. and Guest, D.J. (1977). Scope marking in computer conditionals: A psychological evaluation. International Journal of Man-Machine Studies, 9, 107-118.

Sloboda, J. (1981). Space in music notation. Visible Notation, 15, 86-110.

Sufrin, B. (1982). Formal specification of a display-oriented text editor. Science of Computer Programming, 1, 157-202.

Szlichcinski, K.P. (1979). Telling people how things work. Applied Ergonomics, 10, 2-8.

Van der Veer, G.C. and van de Wolde, G.J.E. (1983). Individual differences and aspects of control flow notations. In T.R.G. Green, S.J. Payne and G.C. van der Veer (eds.), The Psychology of Computer Use. London: Academic Press.

Wright, P. (1977). Presenting technical information: A survey of research findings. Instructional Science, 6, 93-134.

SOFTWARE ENVIRONMENTS

ACTIVE HELP SYSTEMS

Gerhard Fischer, Andreas Lemke and Thomas Schwab

Research Group on Knowledge-based Systems and Human-Computer Communication
Department of Computer Science, University of Stuttgart
Federal Republic of Germany

Good on-line help systems are of crucial importance for the computer systems of the future. An increased functionality (required by the many different tasks which a user wants to do) will lead to an increased complexity. Empirical investigations have shown that on the average only 40% of the functionality of complex computer systems are used. Passive help systems (which require that the user requests help explicitly from the system) are of little use if the user does not know the existence of a system feature. Active help systems should guide and advise an user similar to a knowledgeable colleague or assistant.

1. INTRODUCTION

The purpose of computer-based man-machine systems is to direct the computational power of the digital computer to the use and convenience of man. There has been great progress towards this goal and the dramatic price reduction in hardware has led to totally new possibilities. But other aspects have not kept pace with this progress, especially how easy it is (not only for the expert but also for the novice and the occasional user) to take advantage of the available computational power to use the computer (for a purpose chosen by him/herself[1]).

Most computer users feel that computer systems are unfriendly, not cooperative and that it takes too much time and too much effort to get something done. They feel that they are dependent on specialists, they notice that software is not soft (i.e. the behavior of a system cannot be changed without a major reprogramming of it) and the casual user finds himself in a situation like in instrument flying: he needs lessons (relearning) after he did not use a system for a long time.

Our goal is to create symbiotic, knowledge-based computer support systems which handle **all** of their owner's information-related needs (these needs will be quite different for different groups of users).

Our system building efforts can be characterized with the following metaphor: the user of an interactive system is like a traveler on a modern highway system with (maybe ill-defined) goals, methods, heuristics and decision points; the user support systems serve as tour guides who provide assistance if requested and who point out interesting places (based on knowledge about the traveler and the country-side).

2. USER SUPPORT SYSTEMS

The user-support systems which we envision must be build as knowledge-based systems. First the architecture of knowledge-based systems will be described and then several kinds of user support systems will be briefly characterized.

2.1 Knowledge-based Human-Computer Communication (HCC)

Knowledge-based systems are one promising approach to equip machines with some human communication capabilities. Based on an analysis of human communication processes we have developed the model shown in Figure 2-1.

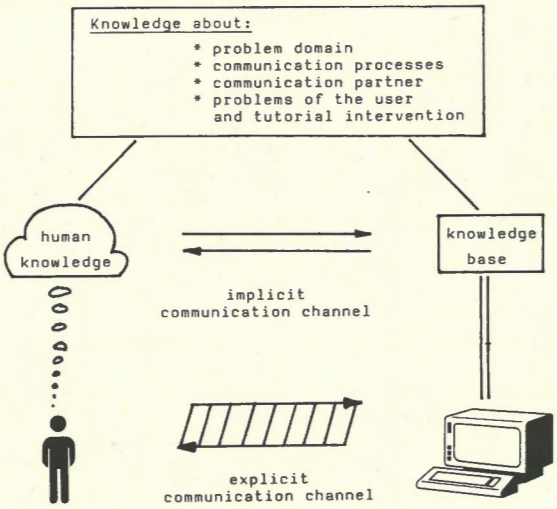


Figure 2-1: Architecture for knowledge-based HCC

The system architecture in Figure 2-1 contains two major improvements compared to traditional approaches:

1. the **explicit** communication channel is widened. Our interfaces use windows with associated menus, pointing devices, color and iconic representations; the screen is used as a design space which can be manipulated directly (see Figure 4-1).

2. information can be exchanged over the **implicit** communication channel. The shared knowledge base eliminates the necessity that all information has to be exchanged explicitly.

The four domains of knowledge shown in Figure 2-1 have the following relevance:

1. **knowledge of the problem domain:** research in Artificial Intelligence has shown that intelligent behavior builds upon large amounts of knowledge about specific domains (which manifests itself in the current research effort surrounding expert systems).

2. **knowledge about communication processes:** the information structures which control the communication should be made explicit, so the user can manipulate it.

3. **knowledge about the communication partner:** the user of a system does not exist; there are many different kinds of users and the requirements of an individual user grows with experience.

4. **knowledge about the most common problems which users have in using a system and about tutorial invention:** this kind of knowledge is required if someone wants to be a good coach or teacher and not only an expert; an user support system should know when to interrupt a user.

Knowledge-based communication allows to replace canned information structures (like they were used in traditional CAI) with dynamically generated information based on the contents of the underlying knowledge base.

2.2 Tutorial systems

Tutorial (or instructional) system should assist a learner; the material is presented based on psychological and pedagogical considerations. The learner (not familiar with the concepts and with the structure of the system) is unable to articulate his goals. The interaction is determined to a large extent by the system and therefore the information can be structured in advance. A (not too sophisticated) example of this sort of system is the LEARN system embedded in UNIX. In our work we try to enhance tutorial systems with modern HCC techniques, e.g. the dynamic generation of visual representations (see Figure 4-1; (Nieper 83)).

2.3 Explanation Systems

Information processing using the implicit communication channel (see Figure 2-1) implies that a system infers information, makes assumptions and draws complex conclusions. Explanation systems should provide insight into complex computations. The user has communicated a goal to the system and he wants to have feedback what the system has done. In a system which makes extensive use of defaults (e.g. the document preparation system SCRIBE) it is important that the user can ask the system to explain which default assumptions exist and how they contribute to a certain outcome.

2.4 Documentation Systems

We consider a documentation system as the kernel of a software production environment. It contains all the available knowledge about the system combined with a set of tools useful to acquire, store, maintain and using this knowledge. It serves as the communication medium between clients, designers and users throughout the entire design process. A valid and consistent documentation during the programming process itself is of special importance in incremental design processes to provide answers to the questions: what has been done, what is the next thing to do, how does the current implementation compare with the specifications, etc.. DOXY (Lemke, Schwab 83) is a prototypical implementation of our general approach (Fischer, Schneider 84) to build computer-supported program documentation systems.

2.5 Help Systems

Help systems should assist the user in specific situations. The user knows his goals but he cannot communicate them to the system (passive help systems) or he does not know that the system offers better ways to achieve the task (active help systems). Contrary to tutorial systems help systems cannot be structured in advance but must "understand" the specific contexts in which the user asks or needs help. The interaction is much more initiated by the user (in passive help system) or mixed-initiative (in active help systems) than in tutorial systems.

In the remaining part of the paper only help systems will be discussed and our system building efforts in constructing them will be described.

3. HELP SYSTEMS

An increased functionality (required by the many different tasks which a user wants to do) will lead to an increased complexity. Empirical investigations have shown that on the average only 40% of the functionality of complex computer systems are used. Figure 3-1 is based on our empirical investigations through careful observations (e.g. persons using systems like UNIX, EMACS etc. in our environment) and describes different levels of system usage which is characteristic for many complex systems. The different domains correspond to the following:

D1: the subset of concepts (and their associated commands) which the user knows and uses without any problems.

D2: the subset of concepts which he uses only occasionally. He does not know details about them and he is not too sure about their effects. A passive help system can help him to take advantage of the commands in D2.

D3: the mental model (Fischer 83) of the user, i.e. the set of concepts which he thinks exist in the system.

D4: D4 represents the actual system. Passive help systems are of little use for the subset of D4 which is not contained in D3, because the user does not know about the existence of a system feature. Active help systems which advise and guide an user similar to a knowledgeable colleague or assistant are required that the user can incrementally extend his knowledge to cover D4.

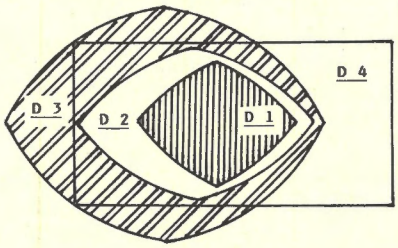


Figure 3-1: Different levels of system usage

Based on these findings it is obvious that a good help system must have a passive and an active component. Nearly all existing help systems are passive help systems where the user takes the active part and the system responds to his requests. Possible interaction techniques are menus, keywords and natural language. Some help systems are built as browsers in a large network of descriptive information where the user moves around and searches answers to his problems.

3.1 Passive Help Systems

3.1.1 Keyword based Help Systems

Assuming a user knows the name of a command but not its details, keyword based help systems are a quick, easy to implement and sufficiently reliable choice. Many of these help systems still fail when the user does not know the exact name. The success rate can be improved by adding synonym lists and pattern matching capabilities which make is possible to express problems in different ways.

Keyword based help systems can be used for: explanation of terms, description of commands and function keys, and search for related concepts. They are of little help clarifying general topics where it is difficult or impossible to describe the needed information with a single word. Existing keyword help systems are static and do not take the user's special situation or his level of expertise into account. Therefore in many situations too much information is provided.

3.1.2 Natural Language Based Help Systems

Natural language provides a wider communication channel between the user and the system than menu selection or keyword based interaction. Observation of human "help systems" suggests that often a dialog rather than a single question and answer is necessary to identify the user's problem or to explain a solution. Natural language based help systems offer the following possibilities:

1. the user has **multiple ways** to formulate a problem. Natural language provides much more flexibility than the best synonym lists.

2. with natural language **failures are soft**. Most of the user's utterances give at least a hint to where the problem lies.

3. **misconceptions** of the user can be identified from his utterance. This gives an important clue for building a user model.

4. the user can not only ask a specific question, but he can **describe** his goals, his intentions and his difficulties.

A problem for novices is to find the appropriate words to describe their problems based on a lack of experience in talking about concepts of the used computer systems.

Current natural language systems implement only a **restricted subset** of natural language. It is an unresolved problem how to describe this subset to the user. It may well be that it is easier for the user to learn a formal way of expressing his needs than learning the restrictions in the natural language interface.

3.2 Active Help Systems

There are many different systems aids which can be considered as active help systems. Canned error messages occurring every time the user does something wrong are the simplest form of an active help system. The active help systems which we envision do not only respond to errors but notice -- based on a model of the user and the task -- **suboptimal actions** of the user. In an operating system the user may issue the sequence of commands:

```
DELETE PARSE.PAS.1
DELETE PARSE.PAS.2
DELETE PARSE.PAS.3
```

leaving the file PARSE.PAS.4. An active help system (Finin 83) might offer the advice to use the command "PURGE PARSE.PAS" which deletes all the old versions of a file.

A metric is necessary to judge how adequate an action of the user is. Except for simple problem domains (e.g. a game where an optimal metric can possibly be defined (Burton, Brown 76)), optimal behavior cannot be uniquely defined. Our actual implementations (see section 5) of help systems are constructed for an editing system and the metric chosen is the number of keystrokes. There is no doubt that this is a very crude and questionable metric. In our future work we will represent more appropriate metrics, e.g. like defining the right relation between cognitive and physical effort.

If a user and a help system rely in their understanding on a very different metric the same difficulty like with human help occurs: **the help system "forces" the user to do something which he does not want to do.** A possible solution to this problem might be to make the metric visible and to allow the user to change it; but we must be aware that this increases the control of the user as well as the complexity of the system and it will be of little use if we do not find adequate communication structures for it.

4. REQUIREMENTS FOR HELP SYSTEMS

This section describes issues which are relevant for both types of systems described in the next section.

4.1 The user as an information processing system

Design guidelines must be based on a thorough understanding of the strength and weaknesses of the human information processing system. In our work we have paid special attention to the following issues:

1. the **power of the visual system** as a very efficient chunking method should be utilized. In Figure 4-1 (Nieper 83) we show a visual representation of a LISP data structure which gets generated automatically from the symbolic representation. Given the following lists

   l1 <== ((eins 1) (zwei 2))
   l2 <== ((drei 3) (vier 4))

   the two operations APPEND and NCONC generate on the surface the same result

   (append l1 l2) => ((eins 1) (zwei 2) (drei 3) (vier 4))
   (nconc l1 l2) => ((eins 1) (zwei 2) (drei 3) (vier 4))

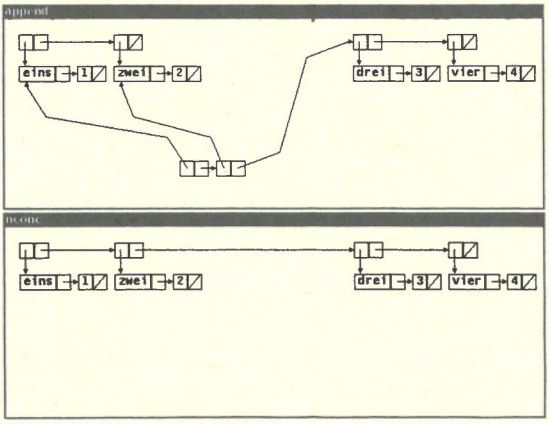


Figure 4-1: Visualization techniques to assist in LISP programming

The visualization in Figure 4-1 shows what really happened: APPEND is a nondestructive operation but uses up two more memory cells whereas NCONC is a destructive operation.

2. **recognition memory** provides different access mechanisms compared to recall memory; therefore menu-based system and property sheets (like in the STAR interface) may be helpful for the novice and the casual user.

3. the **scarce resource** in human-computer systems is not information but human attention; this implies that techniques for intelligent summarizing, prefolding of information etc. have to be developed based on a model of the user.

4. our **short term memory is limited,** i.e. there is a very finite number of things which we can keep in mind at one point of time. Help systems describe and illustrate features about certain aspects of a system; in many cases it is necessary to see both information structures at the same time which requires window systems.

5. our systems must be **consistent and uniform** for the user and consistency must be based on an adequate model of how a system will be used. Help systems are part of the system and therefore they should have the same interface as the other parts of the system. Otherwise the well-known situation will occur that we need *a help systems to use the help system.*

4.2 Modeling the user

Many existing help systems do not take an individual user's special situation or his level of expertise into account. The following knowledge structures have to be represented:

1. the user's conceptual understanding of a system (e.g. in an editor, text may be represented as a sequence of characters separated by linefeeds which implies that a linefeed can be inserted and deleted like any other character).

2. the user's individual set of tasks for which he uses the system (a text editor may be used for such different tasks as writing books or preparing command scripts for an operating system).

3. the user's way of accomplishing domain specific tasks (e.g. does he take full advantage of the systems functionality?).

4. the pieces of advice given and whether the user remembered and accepted them

5. the situations in which the user asked for help.

One main task of an active help system is to monitor the user's behaviour and reason about his goals. Sources for this information are: the user's actions including illegal operations. This is based on the following hypotheses (Norman 82):

> "a user does not make arbitrary errors; all operations are iterations towards a goal."

Classification of users with the help of stereotypes can be used to make assumptions about the expected behaviour of the user (Rich 79).

4.3 Modeling the task domain

Knowledge about the task domain imposes constraints on the number of possible actions. A model of the task domain describes reasonable goals and operations. In UNIX if a user needs more disk space it is in general not an adequate help to advise him

to use the command "rm *"[2] (Wilensky 83).

The user's goals and intentions can be inferred in situations where we understand the correspondance between the system's primitive operations and the concepts of the task domain. If the user of an editor repeatedly deletes characters up to the beginning of a word this can be recognized as the higher level concept delete-beginning-of-word. In ACTIVIST (see section 5.2) these concepts are modeled as plans.

This mapping should be **bidirectional.** Given a problem of the task domain, a help system must be able to indicate how it can be solved using the primitive operations (e.g. in the domain of text editing the help system finds the sequence of operators *delete-region insert-region* for the user's goal of moving a piece of text).

4.4 Help Strategies

Help systems must incorporate tutorial strategies which are based on pedagogical theories, exploiting the knowledge contained in the model of the user. Stragegies embodied in our systems are (Fischer 81):

1. **Take the initiative** when weaknesses of the user become obvious. Not every recognized suboptimal action should lead to a message. Only frequent suboptimal behaviour without the user being aware of it should trigger an action of the system.

2. **Be non-intrusive.** If the user does not accept our suggestions, let him do the task in his way.

3. **Give additional information** which was not explicitly asked for but which is likely to be needed in the near future.

4. Assist the user in the **stepwise extension** of his view of the system. Be sure that basic concepts are well understood. Don't introduce too many new features at once (Fischer 81).

5. PROTOTYPICAL IMPLEMENTATIONS

In this section a passive and an active help system for the editor BISY are described which we have developed and implemented. BISY (Bauer 84) is an EMACS-like, screen oriented editor, also developed in our research group.

BISY was chosen for the following reasons:

* Editing is a task domain which is complex enough but well understood.

* BISY is integrated in our working environment (where we have a wide variety of tools at our disposal: LISP, OBJTALK, a window system etc.). Therefore it was easy to add a help system as an additional feature.

* Editing systems are an often used tool in our work and therefore it is easy to get feedback about the usefulness of our help systems.

The current implementation of the two help systems can only deal with *Cursor movement* and *deletion* tasks. In this domain BISY offers a rich set of operators. In addition to character oriented commands there are higher level operations for words, lines, paragraphs and lists. The systems' level of understanding is limited to these con-

cepts. Concepts of subject domains in which editors are used (e.g. *address* in a letter) are not handled in the current implementation.

5.1 PASSIVIST: An Example for a Passive Help System

PASSIVIST (Lemke 84) is a natural language based help system for the editor BISY. The first step in the design of this system was to get an impression of the real needs of the user. In several informal experiments a human expert simulated the help system in editing sessions with users of different expertise. The results indicated a fairly diverse set of problems ranging from finding keys on the keyboard up to complex formatting tasks.

PASSIVIST provides help to requests like (translated into English):

* How can I get to the end of the line?
* I want to delete the next word.

A passive help system has the choice between the following help modes:

* **Explain:** The system gives hints (e.g. classifies the problem, reminds the user of certain facts) to assist him to solve the problem himself.
* **Solve:** The system provides and explains a complete solution.
* **Demonstrate:** The solution of the user's problem will not only be explained but each step will be demonstrated.
* **Execute:** Sometimes a solution may be too complicated to present to a beginning user. In these cases the system simply does what the user wants to do.

PASSIVIST uses a combination of the first and the third strategy in which each step of the solution is presented and explained to the user who then executes this step and sees immediately the resulting effects (see section 4.1). Help is currently given only as text, generated from slightly abstracted canned information structures. We are planning to augment this by displaying key sequences and subgoals graphically.

5.1.1 A Sample Request to PASSIVIST

Assume the user wants to delete a line. He hits the help key, a help window pops up and he is prompted for a question.[3]

Question: How can I delete the next line?

Recognized words: delete next line
Ignored words: How, can, I, the ?

The system indicates what it has understood. This is a common technique found in human discourses. It allows to correct misconceptions and to build a model of the understanding of the partner. In this case the user can see that the system's understanding is limited.

The system analyzes the sentence, identifies the user's problem and computes a solution by using primitives of the editor which are available to the user. The system first gives a description in natural language (using more or less canned text) and then tells the user the actions which he has to take:

First, move the cursor to the desired line.

Command: cursor-down Key: down-arrow

The user hits the mentioned key and sees the cursor moving down one line while staying in the same column.

The next step of the solution is rather complicated and is therefore explained by the system. The help system has knowledge of certain stereotypical problems and knows where it should give an explanation.

There is no primitive operation to delete a line as a whole. First you have to make it empty and then delete the left or the right line delimiter.

In order to make a line empty, move to its beginning and then delete the rest of the line.

Command: set-cursor-to-beginning-of-line Key: ^A

^A means hold down the **CTRL** *key, hit the* **A** *key and release the* **CTRL** *key.*

It is assumed that the user is a beginner which has no experience with computer key boards. Therefore, for the first two times CTRL-keys are explained.

When the user hits **^A** the system explains the next step which is to remove the rest of the line.

Command: rubout-line-right Key: ^K

^K means hold down the **CTRL** *key, hit the* **K** *key and release the* **CTRL** *key.*

The user hits **^K** and the line becomes empty. The last step is to remove one of the line delimiters.

Command: rubout-line-right Key: ^K

The command **rubout-line-right** *with the cursor at the end of the line merges the current with the following line. This is used to delete the empty current line.*

5.1.2 Implementation

PASSIVIST is implemented in OPS5 (Forgy 81). OPS5 is an interpreter for production systems which has been used in building expert systems.

Flexible parsing using OPS5 is achieved by a rule based bottom up method. The consistent structure of the system as a set of productions and a common working memory allows the use of the same knowledge in several stages of the solution process. For example, knowledge about the state of the editor is not only used to select a possible solution for the user's problem but also to aid to disambiguate the user's utterance.

The german word 'letzt' may mean the 'previous' as well as 'the last in a region'. In the phrase

die letzte zeile (the last line)

with the cursor being at the beginning of the editing buffer it is clear that the user means the last line of the buffer.

Both the model of the user and the model of the editor state are represented as a set of clauses in the working memory of the production system; examples are

a) the system's model of the user (the number indicates the frequency how often the concept was explained to the user):

```
(User KnowsNotationOfKeySequences 2)
(User KnowsNotationOfCtrlKeys 1)
```

b) the system's model of the editor state:

```
(Cursor  ^Word:                 at-beginning
         ^BeginningOfLine:      t
         ^EndOfLine:            nil
         ^InFirstLine:          nil
         ^InLastLine:           t
         ^BeginningOfBuffer:    nil
         ^EndOfBuffer:          nil)
```

To answer a question of the user the system has to do the following:

1. build a model of the editor state
2. read and scan the question
3. parse the question into an internal representation of the user's goals
4. compute solution
5. present and explain solution

The following rule represents the systems knowledge about deleting the end of a line. If there is a goal matching the condition part (the two clauses labeled *<goal>* and *<region>*) and the cursor is not at the end of a line (otherwise another rule triggers) the system proposes the command *rubout-line-right*.

```
(production DeleteEndOfLine
   {<goal>    (Goal delete <side>  ^Active: t)}
   {<region>  (Side <side>  ^Part: end  ^Object: line)}
              (Cursor  ^EndOfLine: nil)
 -->
   (remove <goal> <region>)
   (make Goal issueCommand rubout-line-right  ^Active: t))
```

Sometimes there are better solutions than those found by the system. If there is only a word or a single character between the cursor and the end of the line, other commands (*rubout-word-right, rubout-character-right*) can be used. The system needs a metric to choose an optimal solution (see section 3.2).

5.2 ACTIVIST: an example for an active help system

ACTIVIST (Schwab 84) is an active help system that pays attention to suboptimal user behaviour. It is implemented in FranzLisp and the object-oriented knowledge representation language OBJTALK (Laubsch, Rathke 83).

A user action is considered optimal if it is done with a **minimum number of keystrokes.** The help system can deal with two different kinds of suboptimal behaviour:

1. the user does not know a complex command and uses suboptimal commands to reach a goal (e.g. he deletes a string character by character instead of word by word).

2. the user knows the complex command but does not use the minimal key sequence to issue the command (e.g. he types the command name instead of hitting the corresponding function key[4]).

Like a human observer the help system has four main tasks:

1. to **recognize** what the user is doing or wants to do.

2. to **evaluate** how the user tries to achieve his goal.

3. to **construct a model of the user** based on the results of the evaluation task.

4. to decide (dependent on the information in the model) **when to interrupt** and in which way (tutorial invention).

In ACTIVIST the recognition and evaluation task is delegated to 20 different **plan specialists**. Each one recognizes and evaluates one plan of the problem domain. Such plans are for example "deletion of the next word", "positioning to the end of line", etc..

A plan specialist consists of:

1. an automaton, which matches all the different ways to achieve the plan using the functionality of the editor. Each automaton in the system is independant. The results of a match are the **used editor commands** and the **used keys** to trigger these commands.

2. an expert which knows the optimal plan including the **best editor commands** and the **minimal key sequence** for these commands.

5.2.1 Recognition Task

All automata are active and try to recognize their plans simultaneously. An editor command issued by the user causes a state transition in every automaton. The input for the automata is the command and the buffer state after execution of the command. The buffer state is defined by the position of the cursor with respect to words, lines and the buffer as a whole.

The automaton in Figure 5-1 can recognize the plan **"delete the left part of the current word"**.

Initially the automaton is in one of the states *START* and *WAIT*.

* *START:* the preconditions for the plan are satisfied. In this case the cursor must be in the middle of a word (i.e. the *onw* predicate is true).

* *WAIT*: the plan cannot be immediately executed. The cursor is not at a word.

The transition *WAIT --> START* initiates the *START-ACTION*. The internal memory of used commands and keystrokes is initialized. While the user executes the plan the automaton follows the solid lines and records the used commands and keystrokes. The automaton action *MATCH-ALL* means that the plan is completely recognized and the evaluation task is initiated. For a command which is not part of the plan a default transition (dashed line) leads the automaton to one of the initial states (only depending on the buffer state).

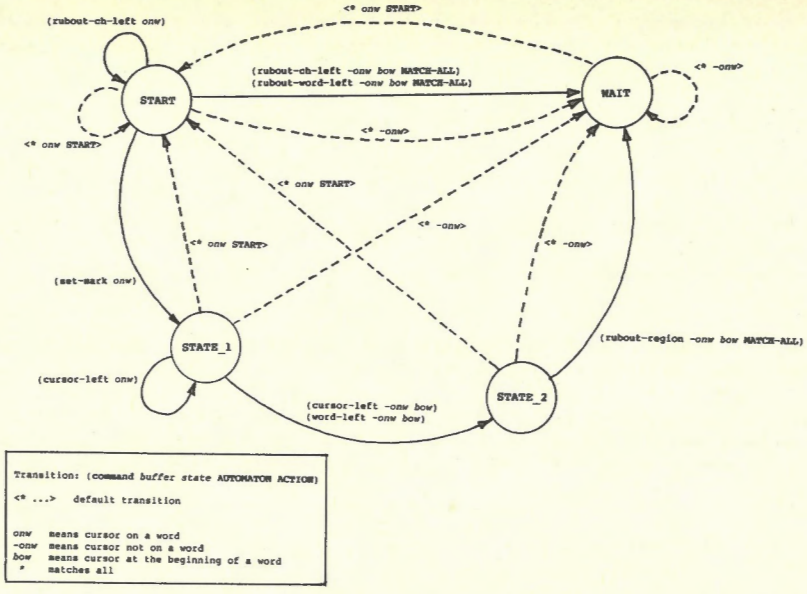


Figure 5-1: Automaton: rubout word left

5.2.2 Evaluation Task

Whenever a plan is recognized by the associated automaton the result of the recognition process is compared with the stored best solution for this plan. Other commands than the proposed ones are considered as a bad solution if the user needs more keystrokes than for the stored solution (e.g. if a word consists exactly of one character the proposed command *rubout-word* is not better than *rubout-character*).

In case he uses the recommended commands his action will only be evaluated as good if he also uses the minimal key sequence.

5.2.3 Modeling the user in ACTIVIST

For each plan there is a knowledge structure which models the user and which contains the following slots:

* *plan-executed* is the number how often the plan was done by the user (in any way).
* *good-done* shows how often the plan was done with the optimal commands and the minimal key sequence.
* *wrong-command-used* shows how often a wrong command was used when the use of the proposed command would have reduced the number of pressed keys. *keysl* counts the unnecessary keys.

* *wrong-keys-used* shows how often the proposed commands were used but not with the minimal key sequence. Here *keys2* counts the unnecessary keys.

* *messages-to-user* shows how often a message concerning this plan was given to the user.

5.2.4 Help strategy

The help strategy of ACTIVIST is variable in the way that all limits are changeable to experiment with different tutorial strategies.

The output of help messages is based on the following global strategy:

* the time between two messages shall be at least *min-message-delay* seconds to prevent information overflow. For a beginner it can be very frustrating to be continuously criticized.

* a message concerning a special plan is given only immediately after the plan was done wrong - not at a later time.

* the message concerning the same error will only be given *max-messages* times to be non-intrusive and to accept that the user wants to do something in his way.

The help activity shall be concentrated more on essential than sporadical errors. Therefore criteria to give a message to the user concerning a special plan are:

1. a plan was done at least *wrong-command-limit* times with the wrong commands and the number of unnecessary keys is greater than *keys1-limit* or

2. the proposed commands are triggered at least *wrong-command-limit* in a sub-optimal way and the number of unnecessary keys is greater than keys2-limit.

A plan which was executed *good-used-limit* times in the optimal way will not be watched any more. The help system assumes that the user is familiar with this feature.

6. FUTURE RESEARCH

Our general research goals to improve human-computer communication with knowledge-based systems have shown that user-support systems in general and help systems specifically are of crucial importance to the success of computer-based systems. Many interesting questions have come up and we will mention the most important ones.

In realistic applications the number of possible user intentions and actions which can be watched simultaneously will quickly reach the limit of the available computational power. Similary to a human tutor the help system is unable to watch all plans simultaneously; therefore it is necessary to concentrate on some plans. Relevant criteria can be based on the following:

1. A plan which was executed in the optimal way several times by the user need not to be observed any more.

2. Very complex plans are not relevant for a novice user.

3. The user can indicate his insecurity in a special domain by questions to the passive help system; these plans can then be observed in detail.

4. The user can decide which plans shall be watched.

Currently our systems work primarily bottom-up, based on data-driven observation of the user behavior. Model-driven predictions (based on a diagnostic model of the most common problems which users have) should be integrated into our system and they could be used to focus the attention of our help systems.

Help systems are currently still constructed as addons to existing systems. Our long ranging vision of how computer systems should be developed is not to write code, but to construct rich knowledge-structures from which we can generate arbitrary projections being used as code, as documentation or as help.

Instead of providing good help systems it may be more fruitful to try to make systems so transparent and so suggestive that there is no need for help at all (e.g. the use of the mouse in our systems has greatly reduced the complexity of many system components; see figure 4-1).

For help systems to be truly useful, the following conditions must hold: the amount to build them must be feasible (e.g. we need support for knowledge acquisition, consistency maintenance, etc.) and the help system itself must be easy to use.

FOOTNOTES

[1]) In the remaining part of the paper we will use the pronoun "he" as a generic notation for a user.
[2]) The command will delete all files in the directory.
[3]) The user input is underlined; system output is in italics.
[4]) In BISY a command can be bound to a function key.

REFERENCES

Bauer, J. (1984). BISY. A Window-Based Screen-Oriented Editor, embedded in ObjTalk and FranzLisp. Institutsbericht, Project INFORM, Institut für Informatik, Universität Stuttgart.

Burton, R.R., Brown, J.S. (1976). A tutoring and student modeling paradigm for gaming environments. Proceedings for the Symposion on Computer Science and Education. Anaheim, California.

Finin, T.W. (1983). Providing Help and Advice in Task Oriented Systems. Proceedings of the Eighth IJCAI.

Fischer, G. (1981). Computational Models of Skill Acquisition Processes. 3rd World Conference on Computers and Education. R. Lewis, D. Tagg (eds). Lausanne.

Fischer, G. (1983). Form und Function von Modellen in Kommunikationsprozessen. Psychologie der Computerbenutzung. H. Schauer, M.J. Tauber (eds). Oldenbourg, Wien.

Fischer, G., Schneider, M. (1984). Knowledge-based Communication Processes in Software Engineering. Proceedings of the 7th International Conference on Software Engineering. Orlando, Florida.

Forgy, C.L. (1981). OPS5 User's Manual. Technical Reports CS-81-135, CMU.

Laubsch, J., Rathke, C. (1983). OBJTALK: Eine Erweiterung von LISP zum objektorientierten Programmieren. Objektorientierte Software- und Hardwarearchitekturen. H. Stoyen, H. Wedekind (eds). Stuttgart.

Lemke, A. (1984). PASSIVIST: Ein passives, natürlichsprachliches Hilfesystem für den bildschirmorientierten Editor BISY. Diplomarbeit Nr. 293. Institut für Informatik, Universität Stuttgart.

Lemke, A., Schwab, T. (1983). DOXY: Computergestützte Dokumentationssysteme. Studienarbeit Nr. 338. Institut für Informatik, Universität Stuttgart.

Nieper, H. (1983). KÄSTLE: Ein graphischer Editor für LISP-Datenstrukturen. Studienarbeit Nr. 347. Institut für Informatik, Universität Stuttgart.

Norman, D.A. (1982). Five Papers on Human-Machine Interaction. CHIP Report 112. University of California, San Diego.

Rich, E. (1979). Building and Exploiting User Models. Ph.D. Thesis. Carnegie-Mellon University.

Schwab, Th. (1984). ACTIVIST: Ein aktives Hilfesystem für den bildschirmorientierten Editor BISY. Diplomarbeit. Institut für Informatik, Universität Stuttgart.

Wilenksky, R. (1983). Talking to Unix in English: An Overview of an On-line UNIX Consultant. Technical Report. Division of Computer Science, University of California, Berkeley.

FATAL ERROR IN PASS ZERO: HOW NOT TO CONFUSE NOVICES

Benedict du Boulay*, Ian Matthew+

*Cognitive Studies Programme, University of Sussex
+Department of Computing Science, University of Aberdeen
United Kingdom

All novice programmers find that their initial programs are rejected by the compiler in a flurry of incomprehensible error messages. Some messages are even hostile (e.g. fatal error in pass zero) and leave the novice sadder and certainly no wiser. The quality of error messages is usually the loser when the compiler writer attempts to balance conflicting design constraints such as size, speed, quality of target code and utility of use by competent programmers.

We believe that novices' programs should be passed through a series of Checkers which are designed to trap and comment on the particular kinds of errors made by them. Such systems may have to make several passes through the program, even to provide an apposite comment on a syntactic error. For logic checking such systems will need access to a description (in some form) of what the novice's program is supposed to do. Only when a novice's program passes through all the Checkers successfully should it be submitted to the standard compiler.

This paper surveys existing attempts to build "intelligent" compilers which are considerate of novices' difficulties. It then describes our own progress towards the construction of program Checkers for use by undergraduates learning Pascal.

1. INTRODUCTION

Pascal is widely used in the initial teaching of computing science in universities. Despite its many good qualities, it has a number of drawbacks as a first programming language, over and above specific technical ambiguities and deficiencies which do not concern us here, see e.g. Welsh et al. (1977). The main difficulties are the large amount of detail which has to be mastered to make even a simple first program work, the lack of debugging facilities so that a student has to laboriously insert and later remove "write" statements to produce a trace and the fact that the language is compiled rather than interpreted which makes meaningful error reporting more difficult. These difficulties mean that special efforts are needed to set up a programming environment based on Pascal (or another similar language) which is properly tuned to the needs of novices.

The student's crucial experience of computing occurs when she first sits at a terminal and, having mastered the login sequence and the editor, attempts to compile her first Pascal program. The result is usually predictable enough - a screenful of technical mumbo-jumbo whose upshot is that the compiler cannot complete its task. A compiler which we have used with first year students reacts (mea culpa) to very poorly constructed programs with "Fatal error in pass zero". Every year it is necessary to explain this message to the students who find it thoroughly disconcerting. They wonder what 'pass zero' is and why it has a 'fatal' error, how fatal is 'fatal' and what should they do next. Of course the compiler also emits more specific messages but these are often either unintelligible, or worse, positively misleading. Some nice examples of the general problem with compiler error messages in Pascal are given by Brown (1983).

Why cannot these messages be made clearer, less confusing and more accurate? If it were simply a matter of adjusting the wording there would be no problem. The central issue is that most compilers are built for one purpose but are often used for another. They are designed to produce efficient object code in an efficient manner and to emit error messages only as a by-product of this process. This is exactly what is needed by the experienced or semi-experienced programmer who can cope with, and often welcomes, terse diagnostics. For the novice programmer the emphasis is back to front. As far as she is concerned the primary output of the compiler consists of the error messages, and the object code is of only secondary importance. A novice is not usually writing a program because she is interested in the task that it does but because she is learning about the language and its facilities. For her the compiler is a teaching device which ideally should be producing a helpful diagnosis of what is wrong with her program.

Many students are quite unable to relate the messages from the compiler to mistakes which they have made. This arises partly because of their lack of knowledge and partly because the messages themselves often focus on the wrong issue. For example, if a student fails to close off a comment properly early on in a program, many compilers will complain only when they reach the end of the program about still being inside a comment (at best) or report something like "unexpected end of program encountered". In both cases this focuses attention at the end of the program rather than on the offending comment. What the novice needs to know is that the comment which started on line 5, say, was not properly closed off because she typed "*)" or "]", say, instead of "*)" or "}" at the end of that line.

Now clearly it is foolish to expect an all-purpose bug detector and analyser, but there are a great number of errors at the lexical, syntactic, semantic and even logical level that are amenable to some form of automatic analysis and report, so making the novice very much more able to proceed without involving outside help.

The syntactic analysis stage of compiling can be carried out rapidly and efficiently in a single scan of the program text with very little lookahead. A failure to parse at a point is usually reported as an error at that point. Error recovery techniques are employed to enable the compiler to continue its analysis past this point, often achieved by jumping to the next "safe" place in the program and continuing from there. It is usually thought desirable to detect and report as many of the errors as possible on each call of the compiler. This is not necessarily the best policy with novices, especially working at vdu terminals, because there is often too much information on the screen for them to cope with. Our own informal observation indicates that novices often deal only with first one or two errors, moving the other error messages off the screen in the process, and then re-compile to re-generate the remaining error messages which are dealt with in a similar fashion.

There is a sharp contrast between determining the point at which the parse fails and the cause of the error. .By "cause" we mean the mismatch between what the novice typed and what she intended. Very often a mistake or mis-typing early in a program is not actually illegal at the point at which it occurs and its effect lurks insidiously to cause a parse failure much later on, e.g. ommitting to declare a variable. A tutor shown a program which will not compile will try to explain what the student did wrong to cause the failure. Doing this requires a knowledge of the student's intention and her likely state of knowledge. For example, an appropriate comment for the error

```
fori:=1 to 10 do
```

concerns the lack of a space between the "for" and the "i", but it is an unusual compiler which will make such an observation. For example, one compiler gives the following for the above error:

```
errors in pascal program
   6   illegal symbol
           4       4
  59   error in variable
           5
 103   identifier is not of appropriate class
           5
 104   identifier not declared
           4       5
```

One of the issues here is that making sense of the error requires one to look a little beyond the point where the parse fails. It then becomes clear that a space has been missed, rather than, as the above compiler assumes, the earlier omission of a declaration of variable "fori". In general searching for the cause of an error may require multiple scans of the program text in an attempt to determine a best-fit diagnosis, may require a greater degree of lookahead than normal and will require knowledge of common mistakes made by novices, such as in (Ripley & Druseikis, 1978; Kahney & Eisenstadt, 1982; Soloway & Ehrlich, 1982) and, in the case of logical mistakes, information about what the program was intended to do.

Because compilers are not well suited to the task of reporting errors in a meaningful way, we argue that for novices error checking should be undertaken by a system separate from the compiler. Only if the error checker fails to detect any errors should the novice then submit the program to a compiler for translation into object code. The advantage of this method is that it allows each kind of tool to be optimised for its particular job, obviates the need to tinker with existing compilers and provides a gentle transition to more standard working methods for the person who finds the separate error checking phase redundant.

An alternative method involves the use of a structure editor, e.g. the Cornell Program Synthesizer (Teitelbaum & Reps, 1981). Such systems emphasise that a program is a structure rather than just a piece of text by only accepting, and even prompting for, syntactically legal code. The advantage is that the novice is alerted at the earliest possible moment about syntactic and lexical errors. The Cornell system also provides various debugging and tracing aids, allows partial execution of incomplete programs and makes good use of the terminal screen to make it a very powerful environment in which to develop programs. A disadvantage is that jumping to conclusions immediately that a mistake is discovered can lead a system to mis-diagnose what the novice intended and so possibly output an unhelpful error message. It is also unclear how well a novice who has learned on such a system will transfer to a less well endowed programming environment. A related question concerns the comparative educational merit of preventing the novice building a syntactically illegal program as against explaining what is wrong if she does so.

Some people argue that providing more extensive error checking is a bad idea, that novices should be made to do more desk checking and that too ready access to interactive terminals and intelligent aids leads to inefficient program development. They recall their own hard apprenticeship - editing paper-tape by hand, working from binary core-dumps or resoldering the program (the reader can supply her own favourite example). There are two difficulties with this argument. First, old-fashioned methods of development e.g via batch or coding sheets made learning harder because they made the write-test-debug cycle much slower so delaying feedback to the learner. Second, novices often cannot desk check precisely because they are novices and they need the error messages to tell them what is wrong.

In the next section we describe various attempts that have been made to make the learning process easier, concentrating mainly on systems for teaching programming automatically. Thereafter we describe our own preliminary attempt at an error checker for Pascal.

2. A TUTORIAL ENVIRONMENT FOR NOVICES

One method of helping the novice is to integrate the language, the teaching documents, the tutorial help and all other aspects of the teaching environment. Where the language itself is not constrained by a Standard, changes can be made to render it more easily learnable, by, for example, changing names of keywords, the syntax or even its semantics to make the language conform to some overall teaching view. The development of Logo at Edinburgh or Solo at the Open University are examples of this all-embracing approach (du Boulay et al., 1981). While the Open University solution (Eisenstadt, 1982), involving close analysis of students' experiences followed by revision of teaching materials, teaching methods and the language is admirable, there is much less room for manoeuvre with Pascal.

Some people have advocated a completely automated system in the hope that each learner would receive an individually tailored course of instruction. Several systems, for a variety of languages and focussing on different aspects of the programming process, have attempted to provide this. Koffman and Blount's (1975) system, Malt, was able to generate problems and check student solutions for simple assembly language programs. However their method of subdividing the problem into small steps and then checking the student's solution for each step against all possible (generated) solutions does not lend itself easily to high level languages such as Pascal. One of the most versatile tutorial environments was BIP, designed to teach Basic (Barr et al., 1976). It was able to give relatively meaningful error messages, illustrate the action of a student's program pictorially, perform limited checks on a student's answer and select a subsequent problem for the student to solve by reference to a student model, a task difficulty model and a teaching strategy. Our preferance would be for more stringent automatic checks of the student's program and more human involvement in the choice of what program the student worked on. Indeed their own experiment demonstrated no particular advantage for the individually chosen sequence of problems over a pre-set sequence chosen to suite a notional average student.

Both Gentner's (1979) system, Coach, to teach Flow, and Miller's (1978) system, Spade, to teach Logo monitored the student as she developed a program. Indeed the former system attempted to understand the significance of every keystroke, including mis-typings. This involved the hard problem of inferring the student's plan from an incomplete and growing program, a problem which has to be faced by any checking system which does not have the full text of the program available.

Some work has been done on detecting and repairing logical errors in student's simple programs. Goldstein's (1975) system, Mycroft, could detect and repair errors in Logo programs designed to produce line drawings. It made the program conform to a set of assertions about the geometric properties of the drawing it was supposed to produce. It was able to relate inconsistencies between a program and the assertions via a theory of program planning and of errors in plans and so repair the program. Using related methods, Lukey (1980) has shown how certain errors can be eliminated from a Pascal program by sub-dividing the program into smaller logical chunks. His system, Pudsy, was able to deal with various logical errors which could be detected without reference to a specification for the program. It was also able to derive assertions about a program and match these against a specification, also represented as assertions. By detecting mis-matches and then changing the program to eliminate them, it was able to remove further logical errors.

By contrast, Adam and Laurent's (1980) system, Laura, worked not from a set of assertions but from a specimen answer also expressed as a program. Having converted both the specimen answer and the student's program into graphs, the system successively transformed each graph without disturbing the overall effect of the program it represented. Eventually the two graphs could either be shown to match or their irreconcilable differences used to localise portions of the student's program

which probably contained errors. In some cases the system could pinpoint an error exactly, such as the use of a constant of incorrect value. Both methods, i.e. using assertions and working from a specimen answer, have advantages especially where a system is only intended to suggest plausible errors and is not expected to undertake the error repairs for itself.

MENO-II is a program currently under development which detects and comments on novices' semantic (problem-independent) and logical (problem-dependent) errors in their Pascal programs (Soloway et al.,1982). It is used within the context of a programming course so it is possible for the Tutors to provide it with a specification of the problems which the students are working on.

The system works in two phases, first finding errors, then inferring possible misconceptions underlying those errors. In the first phase the student's program is parsed into a deep structure which represents the functional characteristics of the program. Various statements within this representation are annotated with tags indicating their roles. For example, the statement

```
sum := sum + new;
```

is tagged with "running total assignment". A second level of annotation then occurs in which the system attempts to add further tags indicating sub-plans (or "programming cliches"). Finally the system matches the fully annotated representation against a taxonomy of known student planning errors, such as omitting an initialisation step or employing sub-plans in the wrong order. This taxonomy has been derived from an extensive analysis of students' errors. The system then enters its second phase where it attempts to pinpoint the misconception underlying each of the errors detected. Each error is linked to a set of possible misconceptions via a newtwork. At present the system reports the error and also reports all the possible misconceptions to the student. The next version of the system is intended to question the student in an attempt to reduce the number of hypotheses about the causes of the error.

The system has been tried with real students but with mixed success. One difficulty was the enormous range of errors and misconceptions associated with even a single program, with the result that their system could only successfully recognise a small proportion of the errors which occurred. Where the novice made many errors the system had no way to integrate its knowledge of errors into a more global view so that it could report sensibly on a student's failure to coordinate sub-plans in widely separated parts of the program. Nevertheless MENO-II stands as a most interesting example of the incorporation of empirical data into a program for use by novices.

3. A PROTOTYPE ERROR CHECKER

As an initial attempt at exploring some of the issues involved, we have implemented a system designed to comment on certain commonly occurring errors. We present this system not as a model of how this task should be carried out but to underline our argument that the task should be done. The program is implemented in Prolog and works, at present, only for a tiny subset of Pascal. This includes integer, boolean and array types, for loops, conditionals, write statements and assignments. It contains five sub-systems, concerned with lexical, syntactic, semantic and logical analysis respecively, plus a simple trace mechanism. Only the syntactic and semantic sub-systems are described in this paper.

The system reports on a single error at a time and the novice is expected to put that error right before the analysis proceeds any further. This might be regarded as a slightly inefficient way to proceed but it minimises the load on the novice who

can then deal with each error as it comes, as in an interpreted language.

3.1 Syntactic Level

The parser is implemented using two sets of rules. One embodies the correct syntax of Pascal. The other, a set of "mal-rules", corresponds to particular syntactic mistakes committed by novices e.g. confusing "=" (equality) with ":=" (assignment) or ".." (subrange) with "to" (keyword) as shown in the following error message produced by our system. Certainly the wording of the message itself can be improved to simplify the language used and give a reference to a more suitable text. However it is the focus of the message rather than its actual wording which is the main issue here:

```
This message refers to the token "=" on line 4 of the program
contained in file test.p .


2 var i : integer;
3 begin
4     for i = 1 to 5 do
5         write(i)
6 end.

This token is the equality operator. It looks as if you have
confused it with the assignment operator ":=" which was expected
at this point in the for statement between the initial value
expression and the control variable.
Ref: Pascal User Manual & Report 2nd Ed. p23 K. Jensen, N.
Wirth.
```

The system is implemented using the "grammar-rule" formalism in Prolog (Clocksin & Mellish, 1981). In this representation each statement in the grammar of Pascal is translated into an equivalent Prolog rule. The collection of rules constitute a Prolog program which then runs as a parser. It works top-down, left to right, backtracking when a wrong choice of which rule to apply has been made. Mal-rules can be represented the same way and incorporated into the parser, with the proviso that the system attempts a parse using the legal rules before trying any of the mal-rules appropriate at that point. One advantage of this formalism is that it allows the system to grow in a reasonably modular way, adding new mal-rules as developed.

```
/* SOME RULES ABOUT FOR STATEMENTS */

/*1*/ for_statement      -->
        ['for'],
        for_stat_body.

/*2*/ for_stat_body      -->
        [Controlvar],   { variable(Controlvar) } ,
        for_stat_body1,

/*3*/ for_stat_body1     -->
        [':='],
        for_stat_body2.

/*4*/ for_stat_body2     -->
```

```
        expression,
        for_stat_body3.

/*5*/ for_stat_body3    -->
        ['to'],
        for_stat_body4.

/*6*/ for_stat_body3    -->
        ['downto'],
        for_stat_body4.


/* SOME MAL-RULES ABOUT FOR STATEMENTS */

/*7*/ for_stat_body     -->
        [X],    { (res_wrd(X),   err_message(e49,X)) } .

/*8*/ for_stat_body1    -->
        [':','='],      { err_message(e55,':') } .

/*9*/ for_stat_body1    -->
        ['='],  { err_message(e56,'=') } .
```

In the above example the code has been simplified. In particular the arguments to each rule have been omitted. These are used to build up a parse-tree during the parsing process. Rules 1 to 6 are some of the correct rules about 'for' statements. For example, rule 1 says that the statement must start with 'for'. Rule 2 says that this must be followed by a variable. Rules 5 and 6 indicate that either 'to' or 'downto' can be used. Rules 7 to 9 are just 3 of the many mal-rules in the system. Rule 8 looks for the use of ':' and '=' separated by a space, instead of ':='. Rule 9 looks for '=' instead of ':='. When a mal-rule fires it triggers off an error message and halts the parser.

In principle the system could attempt an automatic error repair. This is not a good idea from an educational point of view because it does not bring the mistake to the attention of the learner with the same force. Also if the system makes the wrong assumption about what was intended and carries out an inappropriate repair it may make the situation even more confusing for the novice.

There are two kinds of syntactic error. Those, as above, that have been specifically anticipated and can be recognised by one of the mal-rules, and those that have not. In the latter case the system, at present, comments in a similar manner to other compilers concerning the point at which the error occurred, what token was expected and what token found. In attempting to parse a program, the system first tries all relevant rules from the correct set. Only if they fail does it try the relevant mal-rules. There is thus a difference in emphasis from standard compiling techniques in that the system expects and is designed to cope with the kind of mistakes which novices make. There are some general purpose mal-rules which deal with mis-spelled keywords, omitted spaces or semi-colons, as needed, for instance, in the "fori" example given earlier.

The system sometimes postpones judgement about an error until more information is available rather than, say, dealing with the error at the lexical level even though a violation can be detected at this level. One case concerns the use of comments within a program. Misuse of the comment brackets can cause all kinds of difficulty for the novice, as pointed out earlier. Stripping out comments and checking their legality at the lexical level can make it hard for the system to make an intelligent assessment about what the student intended, and hence what to say to her.

For example, the inadvertent mistyping of "{" for "[" in an array declaration will cause all the rest of the program up to the next instance of "}", if any, to be

treated as a comment.

```
var myarray : array{1..10] of integer;
```

This kind of error can be very difficult to spot especially when the error messages point at it only indirectly. By looking at this error in a syntactic rather than a lexical context the system can make a better guess at what has gone wrong. Both brackets are on the same key on our terminals, but on some one holds down SHIFT to get "{" and others one holds down SHIFT to get "[".

3.2 Semantic Level

Semantic checks are performed on a parsed structure handed over by the syntactic analyser. Only when no further syntactic errors are detected is the semantic analysis stage started. Various checks are carried out including type checks, checks that a loop is executed at least once and that variables are properly initialised before being used (i.e. the user is warned against using default values). Here semantic checks include all those checks that can be carried out without knowledge of the particular purpose of the program. For example, attempting to read from a file before "reseting" it would count as such an error. At present, because of the tiny subset of Pascal used these checks are carried out on the basis of a static analysis of the program.

Both error messages and warnings are output. A warning concerns something which though not actually incorrect (at the given level) may indicate a misapprehension on the part of the novice. A standard format it used for error messages which says what is wrong, shows the section of program containing the error, explains what the novice may have forgotten or been trying to do and points to some auxiliary reading. The following is an example of warning about the use of a "for" loop.

> This message refers to the initialisation of the control variable in the for statement which starts on line 6 and ends on line 7 of the program in file prop.p .
>
> ```
> 4 begin
> 5    i := 1;
> 6    for i := 5 downto i do
> 7        writeln(i)
> 8 end.
> ```
>
> The control variable for this for statement has been used in the final value expression of the same statement. This can cause unpredictable results and was probably not intended.

The problem here is that different compilers treat the initialisation of "i" in the "for" loop differently. The effect is that the loop may either print out 5,4,3,2,1, one number on each line or it may simply print out 5. The novice needs to be warned about the danger.

Misunderstanding the meaning of certain symbols can lead the novice to write syntactically legal but nonsensical code e.g.

```
for i := 1 to 10 do;
        myarray[i] := 0;
```

The intention, partly indicated by the indentation - a factor deliberately ignored by most compilers, was to initialise ten elements of "myarray" to zero. The effect of the ";" after the "do" is to produce ten iterations through a null loop followed by an initialisation of only the tenth element of the array.

4. CONCLUSION

The program implemented so far is simply a small-scale prototype for a much more ambitious system which will include extensive testing of the logic of the novices program against what her tutor wanted. It will also provide a system for tracing and single stepping through a program to assist debugging. At present these sub-systems are implemented only in a most rudimentary form.

As Brown (1983) points out, checking a program at the lexical, syntactic and semantic levels breaks no new computing ground. Part of the difficulty in the past has been that these checks, such as they are, have been regarded as part of the more general effort of building a compiler. Because the compiler writer is trying to satisfy a number of conlicting requirements, there has been a tendency to downgrade the importance of good error messages. Also the need to spot an error at the earliest possible moment makes it hard for the system to reach an intelligent assessment of what the user might have intended and, hence, of what she ought to be told. We believe that novices and experts will be served best, not by trying to make compilers work effectively for both groups by radically improving the nature of their error reporting but by building special purpose error checkers for use by novices which are informed about the kind of mistakes which they make.

REFERENCES

Adam, A. & Laurent, J. (1980), A system to debug student programs, Artificial Intelligence, 15, 75-122.

Barr, A., Beard, M., Atkinson, R.C. (1976), The computer as tutorial laboratory: the Stanford BIP project, International Journal Man-Machine Studies, 8, 567-595.

Brown, P.J. (1983), Error messages: the neglected area of the man/machine interface, Communications of the ACM, 26, 4, 246-249.

Clocksin, W., & Mellish, C. (1981), Programming in Prolog. Springer-Verlag, Berlin.

du Boulay, J.B.H., O'Shea, T., Monk, J. (1981), The black box inside the glass box, International Journal Man-Machine Studies, 14, 237-249.

Eisenstadt, M. (1982), Design features of a friendly software environment for novice programmers, Human Cognition Research Laboratory Technical Report No. 3, Open University.

Gentner, D.R. (1979), Towards an intelligent computer tutor, Procedures for Instructional Systems Development, Academic Press, New York.

Goldstein, I.P. (1975), Summary of Mycroft: a system for understanding simple picture programs, Artificial Intelligence, 6, 249-288.

Welsh, J., Sneeringer, W.J. & Hoare, C.A.R. (1977), Ambiguities and insecurities in Pascal, Software-Practice and Experience, 7, 685-696.

Kahney, H., Eisenstadt, M. (1982), Programmers' mental models of their programming tasks, Proceedings of the conference of the Cognitive Science Society.

Koffman, E.B. & Blount, S.E. (1975), Artificial intelligence and automatic programming in CAI, Artificial Intelligence, 6, 215-234.

Lukey F.J. (1980), Understanding and debugging programs, International Journal Man-Machine Studies, 12, 189-202.

Miller, M.L. (1978), A structured planning and debugging environment for elementary programming, International Journal Man-Machine Studies, 11, 79-95.

Ripley, G.D. & Druseikis, F.C. (1978), A statistical analysis of syntax errors, Computer Languages, 3, 227-240.

Soloway, E., Ehrlich, K. (1982), Tacit programming knowledge, Proceedings of the conference of the Cognitive Science Society.

Soloway, E., Rubin, E., Woolf, B., Bonar, J., Johnson, W. L. (1982), MENO-II: An AI-Based programming Tutor, Research Report No. 258, Department of Computer Science, Yale University.

Teitelbaum, T., & Reps, T. (1981), The Cornell program synthesiser: a syntax-directed programming environment, Communications of the ACM, 24, 9, 563-573.

NOVICES AND LEARNING

ON THE IMPLICATIONS OF USERS' PRIOR KNOWLEDGE FOR HUMAN-COMPUTER INTERACTION

Yvonne Waern

University of Stockholm
Sweden

This paper analyses the situation in which a beginning computer user tries to handle a computer system by only having had a brief period of instruction and manned with a manual. The situation is analysed as a problem solving situation, in which knowledge about how similar tasks are handled outside of the system plays a great role. It is suggested that the following situations will lead to slow learning: when the problem space is great, when necessary methods are difficult to access, when prior methods are inadequate and strong, when prior models are inadequate, and when the problem formulation is misleading. It is further suggested that the following may be learned in this situation: situation specific goal-condition-method rules, higher order rules, problem schemata, and causal explanations.

Empirical observation by means of think aloud protocols and registering of actual interactions are presented. These show that difficulties encountered by beginning users may be interpreted as suggested above. As to the learning content, the observations suggest that beginning users primarily learn situation specific goal-condition-method rules. They may furthermore redefine old or create new problem schemata. Higher order rules and causal explanations were not evident in the data collected.

1. INTRODUCTION

The present development of computer systems creates a particular learning situation. Computer systems are, to a great extent, developed to assist people in performing their ordinary tasks. Office automation systems are developed for office work, computer aided engineering systems are developed for engineers, etc. This means that people who are experts in performing a particular task will have to learn new ways of performing that task. The learning situation can thus be characterised as a transfer situation: prior knowledge of the task and its corresponding methods will be transferred to the computer situation. The question to be posed is thus, how differences in prior knowledge will affect learning computerised tasks. I will present an analysis of this problem as well as some examples of observed learning situations.

I will first focus on the situation in which a person is required to learn the methods relevant to a new computer system by interactions with the system. Of course, the person has to know at least some of the appropriate commands and must have a manual available. This is a rather common situation. After a brief introduction to a system, the user will often have to explore the system alone. Supervisors and computer experts are not always available. In this situation, the person who is going to try to work with the system can be regarded as a problem solver. He has a goal to accomplish and has a number of operations in mind which he would like the system to perform but, since the system is new, the way to arrive at the goal is yet unknown. However, the person in mind has some knowledge about the way the task should be performed, without the system. He knows which general strategy to apply to

different problems and also which particular methods should be used outside of the system. This knowledge can be termed knowledge of the "external" task. The system's way to describe the task can be termed the "internal" task (Moran, 1983). These two tasks may be more or less overlapping. (This was not suggested by Moran, but must certainly be the case, see De Bachtin, 1984). However, the person in question has to learn the internal task, irrespective of the way the external task is represented.

The person's knowledge of the internal task is partial at the beginning of course. The first step in the problem solving attempt consists of using these two sources of knowledge - the external task knowledge and the internal task knowledge - to construct a problem space (Newell & Simon, 1972). The more similar these two types of knowledge are, the more the actual problem space will have in common with well-known problem spaces which have been previously used. Although the problem space should not be regarded as the person's conscious representation of the problem, it will restrict the actual processes during problem solving by defining those representations of concepts and objects and those operations upon these representations which are to be permitted (Newell, 1973).

Next, the person will start a search in the problem space. This means that different operations will be performed on the actual symbols currently available, the results evaluated, and new operations chosen. This search will lead to some result (successful or not) as well as some representation of the procedures used during the search. In the situation analysed here, the representation will include procedures relevant to the task (in the system) as well as procedures which are important for handling the system. This representation includes a "model of the task in the system" as well as a "model of the system", from the person's point of view (Halasz & Moran, 1983). If the task has been accomplished, the model can be called a "success" model. If the person failed to perform the task, the model can be called a "failure" model. It should be noted that these notions of "success" or "failure" models refer to the learning of the system. It can be suggested that when the system has been learned, both successes and failures may be derived from the same model. During learning, a simple failure model may be reflected in an utterance such as: "I thought the system would react to my command in the following way... but it did not".

Failures will, of course, occur during a learning situation and will lead to renewed attempts to attain the goal. In these renewed attempts, different changes can be tried. These changes may reflect a failure model, or may be based on trial and error. There are changes which can take place within the same problem space, by taking a new path from a particular node, but there are also changes which redefine the problem space itself. Such changes can take place by evoking new prior knowledge and new methods with the same problem representation or by reinterpreting the problem. The system model will be built up during these attempts to remediate errors as well as during the successful attempts.

An overview of the different concepts presented here and their relation to the problem situation and problem solution is presented in Figure 1. The figure is not restricted to computer work. I shall try to hold on to generality as long as possible concentrating on the particular problems concerning computer systems when they arise.

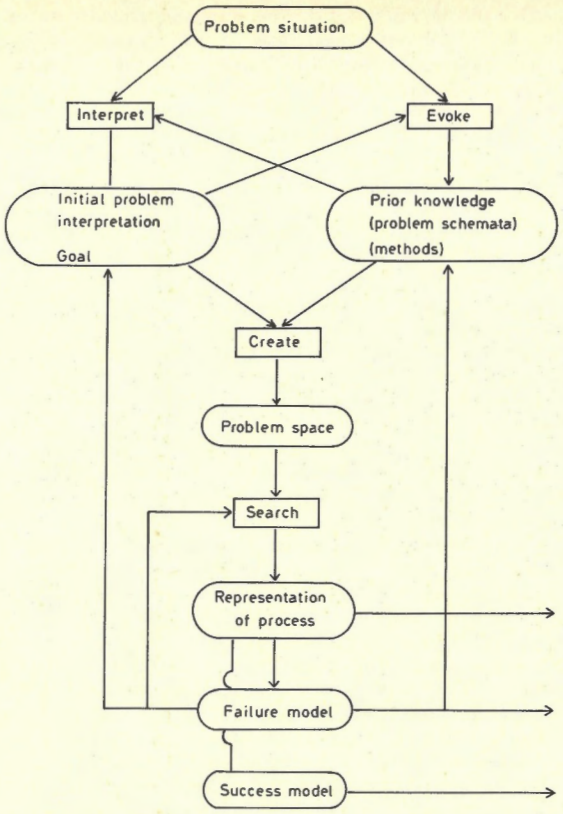


Figure 1 Some interactions between problem interpretation, prior knowledge and problem solving

I shall now propose that some learning is derived from problem solving attempts. (Of course, learning will occur in several different conditions, of which learning by experience is only one. I will however concentrate on learning by experience in this report.) Going back to Figure 1, it is possible to suggest that learning will be quick or slow, depending upon the factors described in Table 1. (Of course, Table 1 should be regarded as a simplification of the problem. There are certainly more varieties in learning speed than quick or slow.)

I shall elaborate on Table 1. By "Size of problem space" is understood the total space of possible operations, which may be performed in the situation known to the problem solver. In a computerised task this space includes the possible commands (or menu choices) as well as the mental acts which may be performed in the situation, such as inferences, decisions, combinations, comparison, planning, etc. It can be said that the smaller the task and the more restricted the system, the smaller the size of the problem space. Whether the user of the system finds a system useful or not, will depend both on ease with which he can learn to use it and on its functionality. If the system performs the task in an easy way, the user will probably find the system useful.

	Learning rate	
	Quick	Slow
Size of problem space	small	large
Accessibility of necessary methods	high	low
Prior methods	functional, strong	disfunctional, strong
Prior models	adequate	nonadequate
Problem interpretation	adequate	misleading

Table 1 Factors which contribute to quick and slow learning respectively

The second point in the table refers to the accessibility of necessary methods. It is self-evident that if the necessary methods are already known, the learning is quick. We would not really refer to such a situation as a learning situation. However, in a new situation several aspects and several methods will have to be learned. The more already known the quicker the total learning. It can also be said, that if old methods are similar to new ones, then learning will also be quick. A more detailed elaboration of the concept will make the statements less trivial. We know that it may be easy to perform a new task, if only one single method has to be learned. It is for instance, easy to learn to log into a system, as long as we do not need to learn anything else. "Accessibility" can thus refer to a situation in which a person can keep a method active in short-term memory as long as is needed to execute it. We know that short-term memory is very restricted. Thus, a method which contains several steps may exceed the capacity of short-term memory. Such a method can then be regarded to be less accessible than a method which fits into short-term memory as a whole. Also, the more new methods a persons has to keep track of simultaneously, the greater the chance that short-term memory will get overloaded and thus that some of the methods will not be readily accessible. When the methods are already well-known, the person can be regarded to have "chunked" them into units, thus saving space in short-term memory. This reasoning therefore implies that a method that can totally be kept in short-term memory is the most accessible. Other methods may vary in accessibility. The cause for this variation lies partially in the "size" of the method: the larger the method, the more to be stored out of short-term memory (in long-term memory or in external memory). Part of the variation lies also in the time needed to access the rest of the method's parts (assuming that one part resides in short-term memory), either from long-term memory or from external memory.

The third influential factor in the table refers to the effect of methods already known and easily accessible. The actual situation may trigger an "old" method, which is easily accessible but not functional in the new context. This might be a situation which is rather common in computerised work, particularly in present day systems, in which the system developers have tried to "simulate" the ordinary work environment. This strategy of system development might encourage the use of methods which are no longer functional or are even disfunctional. The "stronger" these methods are, the more time it will take to suppress or "unlearn" them. By "strong" I mean their probability to get invoked in the particular situation. The "strength" must be regarded to be relative, depending on the characteristics of the situation.

The next factor to be considered concerns the models used in the actual situation. I use the term "model" here to refer to those aspects of the person's conception of the task and system situation which will influence his initial representation of the situation, or the prior knowledge which will be invoked, or the creation of the problem space. Whereas it might be difficult to explain, why one model is influential and another is not, it is easy to agree upon the fact that an adequate model is

helpful (Norman, 1982; Mayer, 1975, 1981), and an inadequate model is harmful (Halasz & Moran, 1982). With the (admittedly vague) definition of the concept of model used here, the model can be regarded as one of the most central components in learning. I will therefore return to an analysis of the model somewhat later.

The last influential factor in Table 1 refers to the interpretation of the problem. In the same way as a model, the interpretation will influence the prior knowledge evoked as well as the problem space created. If the interpretation is misleading, the problem solving will either fail or be inefficient. The effects of misleading interpretations have primarily been pointed out by researchers within the gestalt psychology tradition. Examples include, for instance, perceptual fixations such as believing that the nine-point problem has to be solved within the square formed by the nine points or functional fixation such as not seeing that a pair of pliers may function as a weight in the pendulum problem (Maier, 1931).

The next step in my analysis of problem solving and influential factors in learning by experience, will be concerned with the question of what is learned, in particular in a computer situation.

Since learning always refers to a change relative to something previously existing, we can ask, what kinds of changes can be called "learning". There are several possible changes, which all may be relevant. Learning is often seen as the addition of new facts to old structures. Old structures may also be refined by, for instance, new discriminations or reorganised by new generalisations. Yet another kind of learning results in making procedures more efficient (Anderson, 1982). More efficient means that the procedures are compressed (so that they more easily fit into short-term memory) and that their components can be performed in shorter time. Restructuring changes, in which whole systems of thoughts have to be reconsidered can also be referred to as learning.

Now, what about the content of learning? I will divide learning outcomes from interactions with a computer system into four different categories: goal-condition-method rules, higher order rules, problem schemata and causal explanations. (It should be noted that these categories do not exhaust the possible alternatives. They represent a useful set of learning outcomes with which we can start to analyse what is contained in the concept of "a model of a system".)

It has been suggested by Card, Moran & Newell (1983) and Kieras & Polson (1982 a,b) that learning a computerised system includes learning production rules for performing tasks in the system. These production rules can have the following general form:

```
IF   goal X
AND  condition Y
THEN perform act Z
```

I claim that it is not the form of a production rule, per se, that is interesting (production rules can be used to represent all kinds of learning), but the content itself. In the first kind of learning outcome I will suggest, the production rules learned are concerned with the actual method to be used in an actual situation. I will thus call them "goal-condition-method rules". We can consider the rules as the "atoms" upon which further learning is built. Such a rule would ,for example, include information about the use of simple commands: "if you want the cursor to move forwards and you already have some text written right of the cursor, then type ctrl-f." I have presented some analyses of requirements of different computerised tasks in terms of goal-condition-method rules elsewhere (Waern, 1984).

The second learning outcome I suggest is related to people's attempts to find regularities or construct structures. People try to induce higher-order rules from their specific interactions with the system. Higher order rules concern, for instance, generalisations. The user can reason that if "ctrl" in front of something refers to a character and "meta" refers to a word, then "ctrl" may refer to small entities and "meta" to big ones. The same difference could then apply to lines versus paragraphs.

It has been pointed out that users profit from such internal consistencies and thus can be regarded to learn higher order rules (Barnard, Hammond & Morton, 1981, Reisner 1981 and Payne & Green, 1983). Higher order rules can also concern semantic inferences. If a certain command includes an F and affects something going forward, one might predict that there is a command called B which affects something going backwards.

The third learning outcome refers to the interpretation of the problem given. As we all know, problem descriptions may vary, even though the method to solve the problem is the same. It is evident that people can interpret a particular situation in terms of the concepts relevant to this situation. The concept "schema" was suggested to cover this assimilation of a particular piece of information to a more general structure (Rumelhart & Ortony, 1978). When the situation contains a problem, it is probable that persons learn to interpret that problem in terms of the general methods used to solve problems of that kind (Kintsch & Greeno, 1982). In this case we can say that people learn a "problem schema". Both the problems solved by support of a computerised system and the problems caused by this system can be characterised by more abstract schemata. The schema concept fits into the idea of a "model" very well.

The fourth learning outcome to be suggested concerns different kinds of explanations. People try to explain the reasons why goal-condition-method rules or higher-level rules function as they do. People try to explain why errors have occurred. I will call such explanations "causal explanations". A causal explanation is not necessarily related to procedures, as the action rules and higher-order rules are. For instance, if errors occur when the user types several commands in sequence very quickly, a possible explanation is that the system cannot listen as fast as the user types, or that the system is "busy" working with the first command, and cannot react to the second until the first has been effectuated. This is an explanation which has to do with the properties of the system (in this case inability to split attention).

It may be more difficult to come up with causal explanations than with the other types of learning outcomes. The goal-condition-method rules were related to observable actions and outcomes, as were the higher order rules. The problem schemata were based on the interpretation of problem in terms of possible methods, also an activity based on observable outcomes. However, when trying to find a causal explanation, it is difficult to find an appropriate level of analysis, due to the multilevel character of the computer system. The correct explanations of an error can range from every-day arguments (such as the example above) to descriptions of system design and program listings at hardware level. People cannot be expected to be able to distinguish which level of error has occurred during their first encounters with a particular computer system. The unassisted learner has no possibility (unless he is well acquainted with computer systems) to decide, which levels are plausible, or which explanations belong to which levels. Users with little computer knowledge should try either low-level explanations in terms of goal-condition-method rules or higher-order rules and when these fail, use causal explanations which are based upon every-day reasoning. So for instance they can use different kinds of analogies. The analogy with human behaviour might be obvious (such as the attention explanation suggested above), as well as analogies with other automata (for example "the computer can only do exactly what you tell it to do"). Other analogies might be more task relevant, such as explaining a word processing system in the concepts used for typewriters.

These different contents of learning can all be incorporated in the idea of a "model" of the system. The model of a system will contain action rules as well as higher level rules, problem schemata as well as causal explanations. These rules, schemata, and explanations will be related to successful as well as failing procedures. In the model these different kinds of rules are also related to each other, in smaller fragments or in larger structures, with smaller or larger inconsistencies. The model is succeedingly refined by comparing it to the actual task and system, and the model's inner structure.

The possibility of forming an appropriate model will depend upon the feedback given by the computer system. Not only will the learner have to know that a particular procedure succeeded or failed. He will also have to be able to reason about the success or to diagnose the cause of the failure. This reasoning or diagnosis may be performed by contemplating goal-condition-method rules. However, efficient use of a complex system can never be attained using these low-level rules because of the restricted memory capacities human beings have. The user will need appropriate schemata as well as causal explanations, to access the low-level rules efficiently and to diagnose unexpected errors. We know, that at present, the feedback from most computer systems is quite insufficient for users to form adequate problem schemata or to arrive at causal explanations solely based on their interactions with the system. Beginning users probably do not learn any schemata or causal explanations during their unassisted encounters with systems. Instead, low level goal-condition-method rules will be learned and possibly some higher level rules. A "model" of the system which only contains these types of rules can be regarded to be rather superficial.

The analysis above is based on general psychological principles from human problem solving and learning research, supplemented by a number of characteristics of a computerised task. My next step will be to explain how this analysis applies to a series of observations made on beginning users in different computerised tasks. The aim of this empirical part of the paper is not to test any hypotheses. Its main aim is to illustrate the analysis proposed and to indicate some aspects which need more attention.

2. METHOD

2.1. Subjects

Each subject was totally naive about the particular computer system used. However, all subjects had some (although varied) knowledge about the task to be performed. In each observation, an attempt will be made to characterise the subjects' prior knowledge about the actual task.

2.2. Procedure

The subjects were given a short introduction (spoken as well as written) on handling the system. This introduction was only meant to get them started, not to teach them about the system in depth. The subjects were then asked to perform different tasks. They always had a list of commands (or an overview of menu items) available. The subjects were observed individually. In some tasks, they were asked to verbalise all their thoughts concerned with solving the task. In other tasks, the time taken to perform the task was registered. The actual interactions with the computer system were also registered. The experimenter was passive, except when he asked the subjects to think aloud. The experimenter also intervened when the subject seemed to be quite stuck and could not progress with a particular task.

2.3. Tasks

The following tasks will be presented and analysed:

- diagnosing a magnetic tape,
- drawing inferences from text,
- turning pages,
- debugging a program,
- moving a cursor,
- searching in a database,
- searching in a text

The details of the task will be given in connection with the observations.

3. OBSERVATIONS

In the observations I will try to illustrate the claims made by reference to Table 1. I will thus try to cover the questions of:

- the size of the problem space,
- the accessibility of necessary methods,
- the adequacy of prior knowledge methods,
- the adequacy of prior knowledge models,
- the problem interpretation.

I will also try to analyse the different aspects of what was learned in terms of goal-condition-method rules, higher order rules, problem schemata and causal explanations.

3.1. Size of problem space - diagnosing a magnetic tape

The importance of this aspect can be illustrated by some observations reported by Sääf (1984).

Four subjects were asked to solve the following problem:

> "The computer operations department has given you a magnetic tape. They do not know what it contains. The tape is only useful if it contains a first record with some basic information (record type = G) and some records of either type A or type B. (record type = A or B). They want you to find out if that is the case."

The four subjects were chosen from students enrolled in a 20 week programming course. The topics of the course which are relevant for the observations were: program design according to the JSP-method, COBOL programming and testing. The subjects can be characterised as follows: One was an experienced programmer with 20 years of ICL experience. (Subject 1), two were beginners with some background in computer use (subjects 2 and 3), one was a social scientist, with no computer background at all (subject 4).

There was a marked difference between the subjects in solution times and quality. The totally inexperienced subject (no 4.) worked about 5 minutes on the problem before giving up. The subjects with some computer programming background (nos 2 and 3) solved the problem within 2 and 4 minutes respectively. Both these subjects used JSP-notations (which they just had learned). Their actual solutions were not extented to include all relevant matters. (i.e. the problem could have been treated as a backtracking problem).

Subject 1, the most experienced programmer used three different approaches to solve the task. In his first approach he tried to find an existing program to find out what the tape contained. This attempt took about 10 minutes. After having inquired about the purpose of the experiment, he discovered he was expected to write his own program. He started to construct a flowchart solution, which constituted his second attempt, lasting for about 4 minutes. His third approach started, when the experimenter told him that a JSP-solution was required. He made three different attempts with the JSP-method, each using a different interpretation of the problem. These attempts will be discussed under the heading of "problem schema".

These observations indicate that subject 1 worked in a much larger problem space than subjects 2 and 3 did. Not only did he consider several different methods (looking for an existing program, working wit flow-chart and the JSP-method), but he also considered different kinds of alternatives within the JSP-method, once he was required to use that method. Subjects 2 and 3 had very few alternatives. Since they had just learned the JSP-method, this method was readily accessible and natural to take as a basis for the construction of the problem space. At the same time, their problem solution was less complete than the one offered by the more experienced programmer. This limitation is also probably due to their smaller problem space. One of them (subject 2) considered the backtracking possibility, but did not do any work on that track.

3.2. Accessibility of necessary methods

3.2.1. Drawing inferences from text

The importance of this aspect can be illustrated by some observations reported by Askwall (1984). The subjects were instructed to read texts and draw inferences from these texts. The texts were presented either on paper or on a VDU screen. All subjects worked in both conditions. It was found that the amount of time needed to read the texts did not differ in the two conditions. Nor did the total time used for drawing inferences differ between the conditions. What differed, however, was the methods used to answer the inference questions. In the paper situation, the subjects often sought the information needed to draw the inference among the pages given them. Such searches occurred less frequently in the VDU-situation. Instead the subjects in this situation tried to answer the inference questions by referring to the parts of the text they had remembered. Even though the commands for turning pages were very simple, they still must have presented some degree of memory load. We can conclude that the method to turn pages was less accessible in the VDU-situation than in the paper situation, and was therefore less often used.

3.2.2. Turning pages

The plausibility of the above explanation can be supported by observations in another task (reported in Waern, 1984). Here subjects were simply asked to "turn pages" in a VDU-situation. Three different commands were taught:

(a) go forward n pages;
(b) go backwards n pages;
(c) go to page x.

Eight subjects worked through 192 tasks, requiring them to turn pages. The tasks had either of the following forms:

(a) you are on page 10 and shall move five pages forwards;
(b) you are on page 10 and shall move to page 15;
(c) you are on page 10 and shall move first to page 15, then 3 pages backwards.

The most efficient methods to perform these tasks can be abstractly formulated as follows:

(a) if the task requires you to go a certain number of pages backward or forward, then use the backward or forward command, and fill in the number of pages in the variable;
(b) if the task requires you to go to a certain page, then use the page command and fill in the number of the pages in the variable.

The results (from eight subjects) showed that only one subject used both rules consistently. All other subjects preferred the backward/forward command, even when it was not adequate. Most subjects complained during their inefficient use of commands "that it was so difficult to have to mentally count the number of pages to move". After 96 tasks one subject discovered that she could use the page command (whereas this command should have been used in 48). Her comment was: "How stupid I am: I could use the page command instead!" And so she did in the rest of the tasks, whenever it was more efficient to use this command.

The interpretation can easily be based on the accessibility of the commands. The page command was simply less accessible than the forward/backward commands. It is however, difficult to explain this phenomenon. Was the page command less familiar, and the forward/backward command closer to the methods used when turning pages? (This is the explanation suggested in Waern, 1984). Or was the forward-backward command rule "reinforced" by the tasks, since it could be used in twice as many tasks as the page command? The design does not provide enough information about such questions. The question of the accessibility seems to be important and worthy of further exploration.

3.3. Adequacy of prior methods

3.3.1. Debugging a program

Observations concerning the debugging task have also been presented in Sääf (1984). The same subjects participated in the "tape task" above: one experienced programmer and three inexperienced persons. Here the task consisted of finding four different errors, which were inserted in an existing on line COBOL program. (As mentioned above, the subjects were enrolled in a course, in which among otner things the COBOL programming language was taught).

The subjects used quite different methods at the beginning. The experienced programmer expected to be able to use error messages from the compiler, but the compiler did not indicate any errors. He then started to inspect the program manually. He first checked the overall structure of the program, which was correct. He then went through the program one section after another, comparing the program statements with a program which he himself had developed earlier in the course. Using this method he detected one simple syntactic error (a missing period). The two subjects, who had some knowledge about computers, solved the problem using a method which was quite different. They used the computer to find and correct the errors as they occurred. By this method all four errors were found and corrected by these two subjects, before the first subject had finished his desk-checking. The first subject used the computer method to find and correct the rest of the errors.

These observations suggest that the experienced programmer has learned a method which is no longer adequate, i.e. desk checking. In former days, CPU-time was expensive, and programs could not be checked using the computer. Desk checking was therefore the only feasible method, even if it took a large amount of time and was inefficient. However, the observations also show that the programmer soon adapted to the new situation. He quickly learned to use the more efficient computer method for debugging, even without explicit instruction.

3.3.2. Moving a cursor

The observation which is to be discussed here has been presented in Baladi, (1983) and Waern, (1983, 1984). It is interesting enough to warrant consideration once again.

The observation concerns one subject, a skilled typist, and a small part of a word processing task. In a word processing system the positioning of the cursor plays a central role. The cursor has to be correctly positioned in order to write as well as to delete, to find text strings as well as to justify paragraphs. Here I shall only consider the particular task of moving the cursor. In the word processing system used (a screen editor, called VIDED) the cursor could be moved in several different ways. The arrows keys were most relevant, when small movements were required. The tab, home and return keys could be used for movements over greater distances. The space bar inserted blanks, and thus seemed to move the cursor one step to the right for each short press (repetition was possible by prolonging the time spent pressing a key).

It was found that the subject could quite adequately move the cursor by means of the arrows on the terminal, as long as the cursor was to be moved upwards or downwards or to the left. When the cursor was on the correct line, but to the left of the word to be changed, the subject repeatedly made the same error: She pressed the space bar. In this particular system, the space bar causes blank signs to substitute those signs existing at the place immediately following the cursor. Thus, when the subject pressed the space bar, one or several letters disappeared from the screen. This was of course not difficult to detect. The subject was very annoyed with herself when she observed what she had done. Nevertheless, the next time the same situation occurred - i.e. the cursor was to the left of the target word - the subject pressed the space bar.

This observation can be explained by reference to the subject's prior knowledge. She was a skilled typist. On a typewriter, the space bar moves the typing head forwards (to the right), without affecting the text. In typewriting, pressing the space bar is one of the methods frequently used, to move over existing text. Thus, the observation lets us know that a prior inadequate method can be very difficult to suppress. In particular, it may hold that the method is more difficult to suppress, if it is contained in a higher order goal (here to change a word). The method itself was only subsidiary to the goal. The subject might not have paid enough attention to suppress the inadequate method, when concentrating on changing the word.

3.4. Prior models

3.4.1. Searching in a database

The observations to be presented here have been analysed in greater detail in Linde & Waern (1984). Ten subjects were asked to use a database in order to answer a question. They were asked to identify a person who was described in terms of how he was dressed, what he did at a certain time, and where he was. In the database, there was no person to be found using only this description. Instead, the subjects had to use the description together with the database to make inference about the person who could plausibly correspond to the description. Thus we can say that the database was "incomplete" with respect to the question posed. Observations by means of think-aloud protocols showed that the subjects handled this problem in different ways. Most subjects soon discovered that the database could not answer the question on basis on the description. After this detection, some subjects started to make inferences and guess about possible persons. These subjects quickly arrived at a plausible answer. Other subjects toiled with the idea that the name of the person could be derived from the description, if only the description was used in an intelligent way. These subjects did not start inferring or guessing, after their first attempt failed. Instead, they tried different derivations of the description in order to find the name of the described person. For instance, the description mentioned that

the person was engaged in an activity together with several others. These subjects tried to find an activity in which several other people could be engaged and to identify them. These subjects took much longer to find a plausible "person".

These observations can be interpreted as showing the effect of a model of a database search. The slow subjects may regard a database as a complete source of information, where an adequate search formulation is essential. The quicker subjects soon found out that the database could not answer the question but that they themselves had to make inferences and were allowed to guess. Whereas the slower subjects might have had the idea that the answer had to be "correct", the quicker subjects had the idea that the answer only had to be "plausible". Of course, this interpretation is post hoc. It would therefore be interesting to investigate which conceptions people have of database searches, and how these conceptions affect their search behaviour.

3.4.2. Searching in a text

This observation pertains to a much smaller model than the previous one. In the present observation, the model of how text is represented in a word processing system will be discussed. Different aspects of these observations have also been discussed in Baladi (1983), and Waern (1983), (1984).

Three subjects were given the task to search for a given word. The word was not visible on the screen. In the manual, three different commands were given, in the following order:

(a) search for xxx on the screen;
(b) continued search for xxx on the screen;
(c) search for xxx in the rest of the text;
(d) continued search for xxx in the rest of the text.

All three subjects started with the first command. Nothing happened. One of the subjects (used to computer programming) soon discovered that the third command should be used. The other two subjects (one of whom was totally naive with respect to computers) tried in vain for a long time, until one of them ultimately succeeded. The totally naive subject was told to look at the manual a little closer, after having worked with the problem for 18 minutes. She then commented: "Yes, the commands are all the same". Then the experimenter forced her to use the third command, and the word was found. In a second attempt she solved the same problem, within three minutes.

These observations indicate that the subject's original model of a text (i.e. that a text is continuous) inhibited the discovery of the discrimination the system makes between "text on screen" and "rest of text". The slowest subject did not even detect the difference in the manual! However, we should note that once the difference had been detected the amount the slowest subject learned was drastical. One may conclude that once an adequate model has been found, it will facilitate the handling of the system. This suggestion has also been made by several other researchers (Mayer, 1976, 1981, Norman, 1982). The last two observations and Halasz & Moran (1982) have suggested that an inadequate model will be harmful.

3.5. Problem interpretation - diagnosing a magnetic tape (continuation).

I shall continue to present observations pertaining to the experienced programmer as he attempted a JSP-solution to the tape task. As mentioned above, the subject made three different attempts with the JSP method. The first attempt lasted for about 18 minutes. Here he had difficulties using the JSP-method, but ultimately produced a solution, assuming that record G was the first physical record on the tape. In the second attempt with the JSP-method, he tried to construct a solution in which the first physical record did not have to be record G. He used 11 minutes for this attempt, again having difficulties with JSP-notation and ending up with a JSP-

solution without backtracking. In the last attempt with the JSP-method, he decided to incorporate the backtracking in his solution. So he did, within 12 minutes, ending up with a totally correct JSP-structure.

Two different difficulties in problem interpretation can be identified in this case. The first can be considered to be rather superficial, concerning the interpretation of the concept "first record". One attempt of the solution used the interpretation "first physical record". Next attempt abandoned this interpretation. The second difficulty had a deeper basis, and concerned the consideration of different alternative possibilities in relation to the JSP-method. The problem could easily have been solved without considering the case in which the G, A or B records are missing. (If this case is considered, we are confronted with a backtracking problem in JSP-terminology). The experienced programmer first solved the problem without considering backtracking. This was also the way in which the inexperienced programmers solved the problem. However, the experienced programmer's third and last attempt with the JSP approach used a backtracking solution.

It is probable that this subject's long experience of programming made him more attentive to the possible alternatives. His interpretation of the problem was not as clear-cut as it was for the others, who only considered one single alternative.

3.6. What is learned?

I shall now analyse the contents of learning. The question here will concern, how the observations can best be described. Is it possible to find evidence that subjects learn goal-condition-method rules? That they learn higher-order rules? That they learn problem schemata? That they learn causal explanations?

To answer these questions, we must decide what's meant by these different concepts. I have tried to give an intuitive idea about what they mean above. Now I have to relate these descriptions to actual observations. I will primarily base my interpretations upon the contents of the subjects' think-aloud protocols. These protocols offer some of the subjects own thoughts about their learning. The protocols also provide some information about the subject's intentions, diagnoses of errors and attempts to recover from these errors. This information can give us some hints about the kind of learning that occurs. I shall go through the tasks in the same order as above, and only discuss those observations, which contain anything interesting about learning.

3.6.1. Diagnosing a magnetic tape

This task was described in 3.1. For the subjects who were not experienced programmers, the task can be considered a straight-forward application of a method they had recently learned, i.e. the JSP method. None of the comments made by these subjects had anything to do with new learning. They just tried to recall the JSP-notions. One subject did not recall all of the JSP notions correctly and did not realise he had made a few mistakes. Thus, if the subjects can be considered to have learned anything in this task, the learning content can be described by the following rather backward rule: "If you just have learned a particular method, try to use it in a new task". Such a rule is of course only applicable in a very restricted educational situation, where it often may be quite relevant. We may even assume that the subjects already knew this rule.

The experienced programmer on the other hand had difficulties with this task. I explained the difficulties by referring to his larger problem space. We can thus suppose that what he learned during his three different approaches to the problem was to use the problem space which was required by the experimenter. This can be captured by the following simple rule: "If the teacher requires me to use a certain method, then I have to use it, even though I find other methods easier to use". During his attempts with the JSP-method, he seemed to develop his interpretation of the

problem. He struggled with a particular expression in the problem statement (i.e. "first record") as well as with the problem itself (the backtracking aspect). We can therefore say that the learning here concerned the interpretation of the problem as well as the method learned. The subject's comments repeatedly referred to the difficulties he encountered using the JSP-method instead of one of the other well-known methods (flow-charts, decision tables). These comments indicate that the subject struggled not with simple condition-method rules but with a connected structure of ideas. We can characterise this structure as a schema. Since he already had some well-established schemata, by which the problem could be solved, it was difficult to establish a new one. He ultimately solved the problem, but whether he learned the new schema related to the JSP-method is unknown and cannot be concluded from the data. A schema should be able to incorporate new problems. The data did not contain any new problems.

3.6.2. Turning pages

In this situation (see 3.2.2.), a clear learning effect was found in the time needed to accomplish each task (the average time dropped from 6.0 to 3.8 seconds during 30 trials). As described above, this reduction in time was not attributable to the learning of any new methods. Instead it must be assumed that the subjects refined their procedures to include the chosen methods (even if these methods were inefficient, as was discussed above). This learning can then be referred to as "automation" of procedures, as discussed by Anderson (1982). One of the subjects learned a new method. This was the subject, who discovered that the "page" command could be used in certain tasks. Her comments together with her choice of commands indicate that she used the following rule after having achieved the insight: "If the task requires you to go to a particular page, then use the page command" (see 3.2.2.).

3.6.3. Moving a cursor

This task was presented above (3.3.2.). It can be said, that the subject who had encountered certain difficulties had to learn the following rules: "If you want to move the cursor right, then do not press the space bar" as well as: "If you want to move the cursor right, then press the right arrow key". The subject clearly had difficulties using the rule concerning suppressing certain actions. However, each time she found out what was wrong, she had no difficulties to using the right arrow key. This finding supports the analysis in Table 1: It takes longer to unlearn a strong, dysfunctional method than to learn a single, new one.

3.6.4. Searching in a data base

In this task, (see 3.4.1.), the slower learners showed some amount of learning. This learning has several different aspects. First, the subjects learned that the database cannot give them the information required. This learning is related to the interpretation of the problem or the problem schema. Then the subjects learned that some search commands were more useful than others. Most subjects ended up using a search command based on a time search. Several possible persons could not possible have had time to move from where they were at the time specified to the required place. This learning is rather specific, and can be captured in task specific rules such as: "if you are asked for a person, performing a particular activity at a particular place and a particular time, search for a time close to the mentioned time." and: "if you receive a list of events which happened at approximately a certain time then check which persons could be at this place at that time." This learning may not have been consciously experienced by subjects as such, none of them expressed any conscious choice of key words. The fact that learning had taken place was obvious, both in the way subjects treated newly obtained information (when they started to make inferences on basis of time and place after a while), and in the actual choices of commands the subjects made.

3.6.5. Searching in a text

This task was presented in 3.4.2. It was found that one subject had problems discriminating between the different commands given in the manual.

As already stated, the subject learned this discrimination quickly, once she had noticed the difference. We can say that she learned to attend to the condition sides of the rules: "if on screen, then use command 1, if in rest of text, then use command 3". These rules are quite close to the wording in the manual. Thus it can be assumed that the subject learned to read the manual more carefully than in the beginning. This may be captured by the general rule: "If given a manual, then read every word carefully". It is of course difficult to tell if this general rule was really learned or if the careful reading of the manual was restricted to this particular task. There were no more examples, in which careful reading of the manual could have led to better performance.

In summary, it can be said that the learning outcome found in these examples was mostly concerned with either simple goal-condition-method rules or problem schemata. No examples of higher order rules were found, nor of attempts at causal explanations.

4. DISCUSSION

I have tried to show that common psychological principles can well be used to analyse tasks and interpret observations from beginning users in computerised tasks. Is there anything that makes the study of learning in a new computer system different from the study of learning in other tasks?

Computer system tasks are generally more complex than other commonly used learning tasks. Therefore it can be suggested that the learning of such tasks should also be more complex. Since psychology has suggested different kinds of learning (declarative and procedural learning, event and probabilistic learning, verbal and concept learning, for instance) a complex situation should be expected to contain several different kinds of learning. The observations indicate that users strongly rely upon prior knowledge when approaching a new system. We can thus expect that transfer effects will be strong, positive as well as negative. Depending upon the prior knowledge activated by the task, learning will be quick or slow. Learning processes in computerised situations can be characterised as declarative or procedural, etc., depending upon the prior knowledge activated by the new task.

A second problem in describing learning in a complex task concerns the theoretical representation of the learning content. Many researchers like to talk about subjects' "models" of systems. However, the concept of "model" is still only intuitively defined. I have suggested that at least four different types of learning content have to be included in a "model". My analysis was based upon phenomenological differences between concepts as "actual actions", "abstract rules", "schemata" and "causal explanations". Several other types of learning contents can certainly be suggested. A better coverage of possible alternatives would be arrived at using a suitable taxonomy of learning outcomes. As far as I know, however, such a taxonomy does not exist.

Given the kinds of learning content I have suggested, the observations indicate that some kinds are more common than others in the situations studied here. Examples which could be interpreted as learning of simple goal-condition-method rules and problem schemata were found, whereas examples which could be interpreted as learning higher-order rules or causal explanations were non-evident.

The failure to find the two last kinds of learning can be interpreted in different ways. Either these kinds of learning are rare, and thus did not occur in this small

sample of tasks. Or the particular situation, in which the tasks were studied was not suited to find such kinds of learning. The particular situations all concerned unassisted learning in novices, during their first encounter with rather unknown systems. The learning was characterised as learning by experience, or problem solving. In this kind of learning, the low level rules may be the first ones to be attended to. Also, the problem schema which has been implied was necessary to solve the problem, and we do not know, if it was learned as a schema or as a particular instance. It may be true that higher level rules and causal explanations occur later in the learning process. It may also be true that such types of learning outcomes are easier to achieve when the rules and explanations are suggested by an instructor.

From the observations presented here we may conclude that users' prior knowledge represents a relevant factor when difficulties in the first experience with a computer system are encountered. We can therefore conclude that properties of systems cannot be tested without considering users' prior knowledge.

REFERENCES

Anderson, J.R. (1982). Acquisition of Cognitive Skill. Psychological Review, 89, 369-406.

Askwall, S. (1984). Computer supported reading vs reading text on paper. FOA-report, No. D 53018. Also accepted for publication in International Journal for Man-Machine Studies.

de Bachtin, O. (1984). It is what it's used for. Paper to be presented at the first IFIP conference on Human-Computer Interaction, London.

Baladi, P. (1983). Inlärning av ett ordbehandlingssystem (VIDED). (Learning of a word processing system (VIDED)). B.A. thesis, Department of Psychology, University of Stockholm.

Barnard, P.J., Hammond, N.V., Morton, J and Long, J. (1981). Consistency and compatibility in command languages. International Journal of Man-Machine Studies, 15, 87-134.

Card, S.K., Moran, T.P. and Newell, A. (1983). The Psychology of Human-Computer Interaction. Lawrence Erlbaum, Hillsdale, New Jersey.

Halasz, F. and Moran T.P. (1982). Analogy considered harmful. Proceedings from the CHI '82 Conference: Human Factors in Computing Systems. ACM.

Halasz, F.G. and Moran, T.P. (1983). Mental models and problem solving in using a calculator. Proceedings from the CHI '83 Conference: Human Factors in Computing Systems. ACM.

Kieras, D.E. and Polson, P.G. (1982). An outline of a theory of the user complexity of devices and systems. Project on User complexity of devices and systems. Working Paper No. 1. University of Arizona and University of Colorado.

Kieras, D.E. and Polson, P.G. (1982). An approach to the formal analysis of user complexity. Project on User complexity of devices and systems. Working Paper No. 2. University of Arizona and University of Colorado.

Kintsch, W. and Greeno, J.G. (1982). Understanding and solving word arithmetic problems. Technical Report, Department of Psychology, University of Colorado.

Linde, L. and Waern, Y. (1984). On search in an incomplete database. FOA-report. (in press).

Maier, N.R.F. (1931). Reasoning in humans. II. The solution of a problem and its appearance in consciousness. Journal of Comparative Psychology, 12, 181-194.

Mayer,(1975). Different problem-solving competencies established in learning computer programming with and without meaningful models. Journal of Educational Psychology, 67, 725-734.

Mayer, R.E. (1981). The psychology of how novices learn computer programming. Computing Surveys, 13, 121-139.

Moran, T.P. (1981). The command language grammar: a representation for the user interface of interactive computer systems. International Journal of Man-Machine Studies, 15, 3-50.

Moran, T.P. (1983). Getting into a system: External-internal task mapping analysis. Proceedings form the CHI '83 Conference: Human Factors in computing Systems, The Association for Computing Machinery, 45-49.

Newell, A. (1973). Production system: Models of control structures. Visual information processing. Chase W.G. (Ed.). Academic Press, New York.

Newell, A., Simon H.A. (1972). Human problem solving. Englewood Cliffs, New Jersey.

Norman, D.A. (1982). Five Papers on Human-Machine Interaction. Report NO. CHIP-112-TR-ONR-8205.

Payne S.J., Green, T.R.G. (1983). The user's perception of the interaction language: A two-level model. Proceedings from the CHI '83 Conference: Human Factors in Computing Systems, The Association for Computing Machinery, 202-206.

Reisner, P. (1982). Formal grammar as a tool for analysing ease of use: some fundamental concepts. Human Factors in Computer Systems. Thomas H.J. and Schneider M. (Eds). Ablex.

Rumelhart, D.E., Ortony, A. (1977). The representation of knowledge in memory. Schooling and the acquisition of knowledge. Anderson, R.C., Spiro, R.J. and Montague W.E. (eds). Lawrence Erlbaum. Hillsdale, New York.

Sääf, J. (1984). Can experience be a disadvantage in computer programming. A study of the problem solving approaches of experienced an inexperienced programmers. B.A. thesis. Department of Psychology, University of Stockholm.

Waern, Y (1983). Prior knowledge as obstacle and help in computer aided tasks. Working Papers from the Cognitive seminar, Department of Psychology, University of Stockholm. No. 17.

Waern, Y. (1984). Learning computerised tasks as related to prior task knowledge. Manuscript, submitted for publication.

WEB TEACHING AS A DESIGN CONSIDERATION FOR THE ADAPTIVE PRESENTATION OF TEXTUAL INFORMATION

Piet Kommers

TH Twente, Enschede
Netherlands

The understanding and the successfully acquisition of textual information seems to be highly dependent on the presentation sequence employed. System control and learner control can be optimally combined if there is a conceptual graph which represents the structural relations between the concepts in the text. Primarily based on the notions of Web Teaching (Norman, 1973), one can design an adaptive presentation mechanism. One algorithm for the computation of the centrality index of concepts in a network will be proposed. Consultancy of reader ratings based on prior knowledge seems to be necessary in order to match prior and new information.

1. INTRODUCTION

In this paper, the rational and the global characteristics of an adaptive text presentation system will be presented. Two main antecedents will be proposed:

(a) adaptive instructional systems and their link to theories about human knowledge acquisition;
(b) Knowledge representation and its function in guiding decisions.

The relevance of user oriented text representation can be found in the growing area of VIDEOTEX as found in the current national systems: PRESTEL, ANTIOPE and TELIDON. In the C.A.I. systems we also see a growing number of programs explicitly designed to effectualise a highly individualised growth of knowledge. One important feature of C.A.I. is the advantage if working interactively. This can lead to an accurate correspondence between the information presented by the system, and the user's c.q. pupil's state of knowledge.

As we are only just beginning to know when and how new information is linked into prior knowledge, it seems necessary to resume several theories in cognitive science, and translate them to the specific situation of "interactive consultancy of text base".

By means of computer based text displays, reading can grow into information retrieval. This means that the author is no longer the composer of story-like episodes, but inserts his knowledge elements into a conversational system, which can then be manipulated by the user.

One way to represent the author's knowledge is in the form of a conceptual graph. As we will discuss later on, this graph can be seen as a partially ordered set of concepts, linked by labelled or unlabelled relations. Based on the specific position of a concept in this graph, one can state a certain degree of centrality or decentrality.

In this paper I would like to illustrate the fact that the configuration of concepts derived from expert knowledge, can be of great importance for the control structure

of user guided text presentation. However, before this conceptual graph can be built, we need to converse with experts, about the central and peripheral concepts which should be distinguished. According to several recent theories about knowledge acquisition this dimension (central - peripheral) seems to be quite important, especially when it has been proven to be related to the dimension concrete - abstract. We then have a theoretical framework that can be used to sequential decisions during the presentation of text. Finally we will discuss the problem of consulting the reader's metacognition in order to let him choose the ideal route through a conceptual graph. Referring to recent theories about the "constructive" aspect of knowledge acquisition, we assume that this learner controlled presentation sequence must ideally be coached by a monitoring system component, which adjusts between the actual state of the user's knowledge, and the conceptual graph, derived from the expert knowledge.

2. ADAPTIVE INSTRUCTION AND THE ACQUISITION OF KNOWLEDGE

Instruction can be adapted to the individual characteristics of pupils, if it possesses both possibilities for interaction and some form of intelligence within a specific domain of knowledge. Beyond these two, the instructional medium also needs a strategy from which control decisions can be derived.

One of the strategies often employed in C.A.I. has been derived from the former tradition of learning machines, having been used in the so-called "programmed instruction". Mainly based on skill learning, this strategy says: "Try to perform the next difficult step. If it is not possible, then go back (to the easier level) and try again afterwards." This approach results in a large amount of system control: The system monitors, traces the performance of the learner, anticipates the next steps of the learner, and controls the sequence of information for the student.

As cognitive science developed, differences were noticed between learning processes - whether the learning process was finally directed to mastery of skill or to acquisition of knowledge proved to be important. In the case of verbal learning, based upon conceptual knowledge, it has proven to be quite difficult to unambiguously sequence the subject matter from easy to difficult. The question of how content should be sequenced or ordered has been the subject of educational debates for at least the past 70 years (Dewey, 1902). Nowadays the question of sequencing content matter is certainly quite important, in order to equip presentation devices with some sort of adaptation mechanism.

Task analysis as a sequence of prerequisites seems to be ineffective. The answer to the question "How should verbal information be sequenced ?" can be found in theories of cognitive psychology and artificial intelligence. One of the most essential predecessors in this area can be found in Ausubel (1963), who claims that knowledge can be represented as a cognitive structure. Most of the followers in the Ausubelian tradition elaborate on this basic notion, that knowledge consists of interrelated elements (concepts) which need each other to operate in a mentally correct way. In the last three decades, several investigators have tried to develop models about how knowledge structures change during the process of learning, forgetting, association and authoring.

In the work of Norman and Rumelhart (1973), an interesting strategy to link new knowledge to prior knowledge has been developed (see figure 1). This so called "Web teaching" strategy can be seen as a promising procedure for the sequencing of subject matter: Presentation of new information should match the cognitive organisation a learner already has, and should be derived from the structure in the subject matter. This firstly supposes a rather complete description of the actual state of knowledge in the learner, and secondly a quite detailed representation of the subject matter structure.

Figure 1 How to add new material in memory. Part A and the shaded parts of B and C indicate what is already in a student's memory. If we add new knowledge in the manner indicated by B, it has no interconnections with prior knowledge and it is likely to be difficult to acquire and to retrieve. If the same new knowledge is well integrated with the old, as in C, then learning should be easier and retention better (From: Norman, 1973).

As a consequence of the basic assumption, that knowledge grows by means of interrelation and embedding new elements in prior knowledge, Norman prescribes to begin by relating new topical information to stable topical concepts in the existing knowledge structure. Subtopical information could then be connected to the topical concepts just established, and so forth.

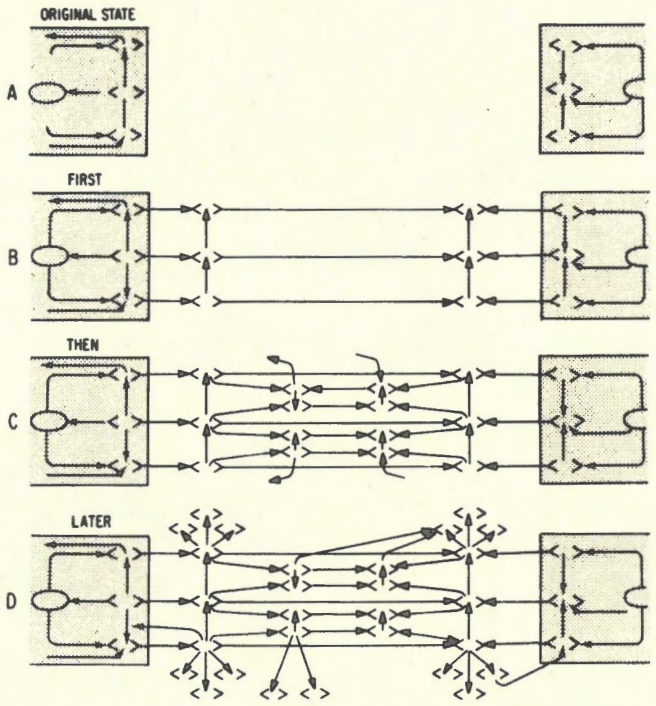


Figure 2 Web learning. First form a net of concepts, each well connected to previous knowledge. Then, slowly build a firm, integrated web onto that net. This procedure is not so easy to perform as that of linear learning, but it should yield superior results (From: Norman, 1973).

In figure 2 the growth of knowledge is expressed. In terms of sequence, it will be clear that Web teaching cannot prescribe the presentation order deterministically. It can however be used as a heuristic paradigm, in order to develop a new mechanism for interactive text presentation.

Adaptive in terms of Web teaching would mean: "According to the actual activated prior knowledge in the reader". As this knowledge is highly individual, variable in time and, as it now appears, too extensive to represent in a proper way, I tend to look at the meta-knowledge of a pupil as a tool for sequence decisions.

In any case there are not enough indications at this time to warrant the assumptions that one ideal sequence exist for every reader, since the ideosyncratic prior knowledge would require strictly individual routes through textual information. On

one hand we have stated that new information has to be adapted to the prior knowledge, on the other hand we have met the necessity of "content structure" as a representation of the domain of new information.

In order to design a computer based program for the adaptive presentation of texts, we are exploring several prototypes which differ in their conversational strategy, but are all based on the notions of knowledge representation as we will describe in 3.

The basic text consists of paragraphs. Each paragraph focussed on one or more concepts, placed in a conceptual network. Each paragraph can be addressed, by means of this network, and then be selected on base of its structural position in the network. During the conversation with the pupil the system must be able to answer the commands given by the student, such as:

(a) give me 4 main paragraphs which deal with the central relations between the following concepts: PRIME MINISTER, QUEEN and LORD MAYOR;
(b) give all the paragraphs in which more than 4 concepts are used.

Beyond the structural position in the network, every concept can be labelled with a centrality factor. This reflects the relative status which is awarded by a domain expert, or an instructional designer. By means of this relevance status we can give the next kind of command:

(c) give me 15 paragraphs which act as descriptors of the 3 most relevant concepts in the network;
(d) give me the three most relevant concepts related in the first order to the concept of e.g. SUFFRAGE.

By means of these procedures, we have the possibility to extract relatively small sub domaines of text, which can serve as key concepts during the transference and acquisition of new information.

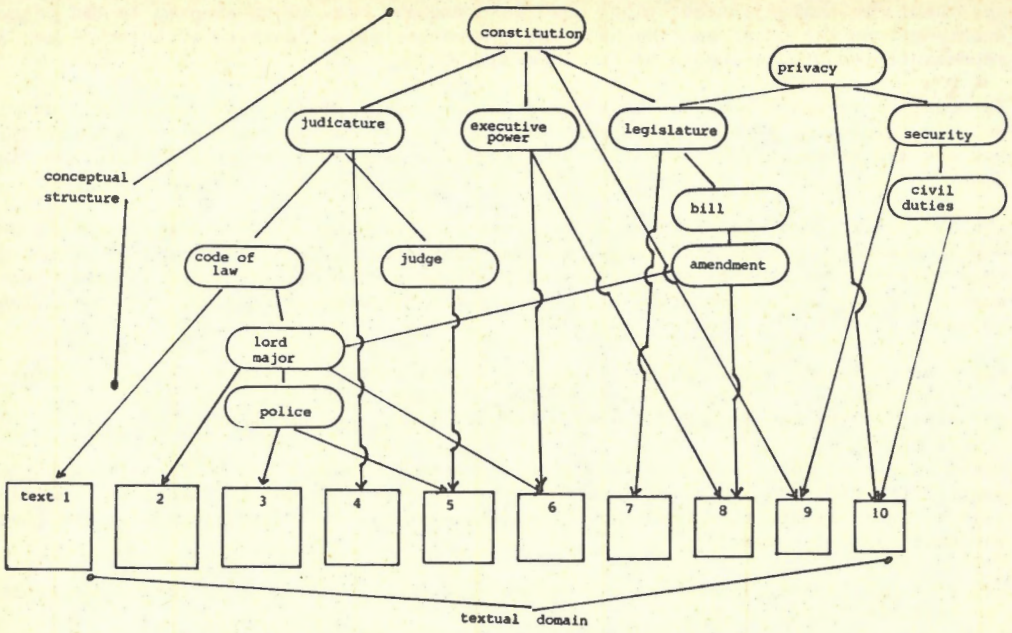


Figure 3 Conceptual network for the control structure during text presentation.

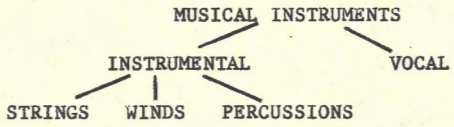
In the example in figure 3 the central part of the "Dutch form of government" could have been constructed by means of several texts, or by means of a direct conversation with an expert on this topic. As can be seen, the concept of CONSTITUTION has a structurally high central position. Some concepts are partly recursively defined, and can finally be described by referring to themselves. Each concept finally points to one or several paragraphs of text, which are visible to the reader. The attainability of paragraphs from a certain point can be derived from the corresponding path in the conceptual network. Algorithms. as has already been stated, can be performed, making use of the numeric label, assigned by the author of the network.

Summarising the previous passage, I would like to stress the relevance of conceptual structures as a directory guide for text presentation. One way to enable the novices to obtain new information is often done by intensifying the relational structure in the text itself. By means of computer based systems however, we have the possibility to control the presentation sequence by means of a paragraph related conceptual network. Based on the notion that the growth of knowledge is highly dependent on the reader's prior knowledge we need a presentation strategy which has been explicitly stated and which can mediate between the conceptual graph as a representation of expert knowledge and the reader's activated state of knowledge.

3. KNOWLEDGE REPRESENTATION AND ITS FUNCTION IN GUIDING DECISIONS

Knowledge representation is often encountered in relation to the frequently used term "Expert systems". On one hand it is commonly remarked that expert behaviour is

more than the availability of knowledge. On the other hand we see a relatively strong accent in one area within A.I. called "Knowledge engineering". In the context of my topic, I would like to restrict the term "Knowledge representation" to the declarative (factual) aspect of knowledge, without the procedural components. As we stated before, adaptivity of a presentation device requires information about the STRUCTURE of the factual elements. This does mean that the complete knowledge of e.g. an expert has to be brought into the system. At first there is a need to represent the conceptual elements which play a role in the knowledge domain. These can consist of short descriptions of concrete objects, persons or attributes. Beyond this they can be entities in a more abstract level: Concepts such as "power", "influence" or " loneliness". One way to generate these conceptual elements is to let experts describe a new domain in a top down way. If we choose a domain such as "musical instruments", we immediately perceive the taxonomic structure:



Of course, after a certain level specification we soon reach causal, temporal and locational relations. Going even further an expert will formulate propositions, ruled... and finally texts.

Before having reached the level of saturation when reading complex texts, network can provide interesting tools for instructional decisions. A first deduction possibility to identify the most central concepts exists. In tree structures the central concepts can be found in the upper nodes. In partially cyclic structures several computations will be needed such as those available in graph theory.

As a try-out we once represented the information in two texts about the Dutch form of government. Five members of a project team notated the most salient relations between them, which were found in the texts. Most parts were different, due to the different levels of detail and completeness which were chosen. In contrast to the conceptual network about musical instruments, a much more complicated structure was found: More cycles and also more diversity in the types of relations between the concepts. At this very moment the six graphs are being analysed - the degree of overlap, and the structural equivalence of different labelled concepts are being calculated. For more details about this method of "construct analysis by the integration of cognitive graphs", see Bakker (1984). Besides the question of how domain experts should be consulted to build rather complete and consistent representations of the content matter, we will be confronted with the question of how to reconstruct the prior knowledge of an individual pupil during the interaction with the presentation system.

One remaining question is the following: to which extend will domain experts and instructional designers be able to construct conceptual graphs as we provisionally did for the area of State Government. Recent research by Tillema (1983) has shown that normal teachers are able to do so after they have been trained for a short period.

Returning to our conceptual graph (which is only partially reproduced here) we can represent some typical sequential states in order to discuss the presentation strategy. The first basic choice we can make is to which extend the system must be equipped with autonomous control strategy, or, to which extend the learner control should be consulted. Because we cannot completely represent the knowledge structure of the learner in the computer, it seems to be practical to rely on a good deal of learner control. The choice repertoire could be reduced to the scope of the plausible alternatives, generated by the system.

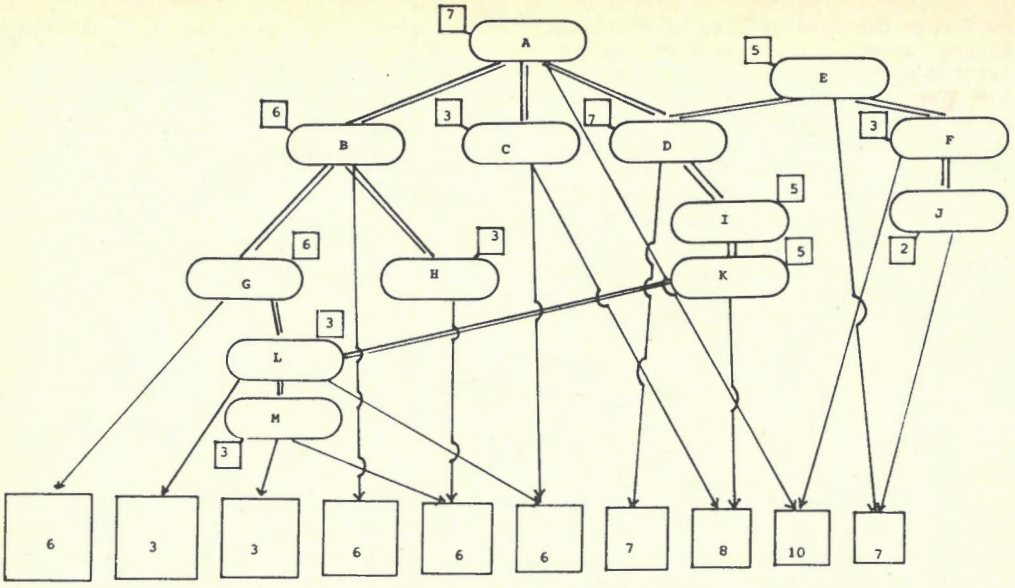


Figure 4 Conceptual graph.

In the graph represented in figure 4 we computed per node a centrality index:

$$\text{centrality of node} = \sum_{j \mid (i,j) \in E} d(j)$$

Each text paragraph can be labelled with an information index: The sum of the centrality indexes of the concepts which point to this paragraph.

For large networks we can make use of the GRADAP package, in order to compare several centrality measures, as described by French (1956), Hubell (1965) and Hoede (1978).

After every paragraph has been given a position on the dimension central-peripheral, the system could start its presentation with a top down strategy as far as the conceptual graph permits it. The crucial point of adaptivity starts as soon as the system perceives two or more equivalent paragraph labels. At this moment the system can display the concurrent concepts and ask the pupil to rank them on the basis of prior knowledge (a high rank will mean that the pupil thinks he is relatively familiar with the concept). By means of this pupil document the conceptual structure will be rescaled until a new concurrent situation requires more choices to be made by the pupil. The ideal numeric impact of the rescaling on the author defined centrality is still a subject which needs some research.

One advantage of the graph-dependent centrality index is that the subject matter expert does not need explicit assumptions about the relative status of the concepts. They become clear after the author mentioned the central relations. Another advantage is the ergonomic plausible representation mode of graphs, which can be quite helpful for authors of didactic texts to reconstruct the basic rationale and eventually modify it.

REFERENCES

Ausubel, D.P. (1963). The psychology of meaningful verbal learning: An introduction to school learning. Grune and Stratton, New York.

Bakker, R.R.(1984). Construct analysis as a method of knowledge integration of cognitive graphs. Memorandum 463, Department of Applied Mathematics, TH Twente, Enschede.

Findler, N.V.(1979). Associative Networks, representation and use of knowledge by computers. Academic Press, London.

Kommers, P.A.M.(1980). Structuring and presentation of texts. Internal report, SVO BS 445, Utrecht.

Kommers, P.A.M., Beukhof, G., Tillema, H.(1982). Design of an adaptive program for the presentation of textual information. Swets & Zeitlinger, Lisse.

Norman, D.A.(1973). Memory knowledge and the answering of questions. Contemporary issues in cognitive Psychology. Solso, R. (Ed.). Winston, New York.

Pask, G.(1976). Conversational techniques in the study and practice of education. British Journal of Educational Psychology, 46, 12-25.

Posner, J.P., Strike, K.A.(1976). A categorisation scheme for principles of sequencing content. Review of Educational Research, 46, 665-690.

Sowa, J.F.(1984). Conceptual structures, information processing in mind and machine. Addison-Wesley, Amsterdam.

Stokman, F.N., Veen, F.J.A.M. van(1981). Gradap, Graph definition and analysis package, volume 1 and 2. Universiteit van Amsterdam.

THE COMPUTER IN THE CLASSROOM

Gerrit van der Veer, Jos Beishuizen

Vrije Universiteit, Amsterdam
Netherlands

The project that will be described in this paper focussed on three topics:
(a) the extent to which the computer may be used to facilitate the adaptation of the learning process to the individual differences found between pupils;
(b) the impact of the computer on the learning environment as an enrichment, especially because of the opportunities it can provide for problem solving behaviour;
(c) the possibilities of the computer for training cognitive skills in the classroom situation.

1. HISTORIC OVERVIEW AND OBJECTIVES OF THE CAI-PROJECT AT AMSTERDAM

At the Vrije Universiteit in Amsterdam a long term project started about 15 years ago. In the few first years, research concentrated on theoretical questions about the interaction between pupil's aptitudes and teaching strategies and the way in which this influenced computer use. Laboratory experiments were designed in which issues such as mathematical learning models,problem solving strategies,learner control or tutor control of the sequence of units, were evaluated. The subject matter in these experiments was partly of an artificial nature. Learning conditions were carefully manipulated. In most cases the pupil interacted with the computer,with an experimenter at hand to help if anything went wrong. On these occasions the learning material often consisted of complete tutorial courses in which the computer took over the role of the teacher by explaining new subject matter or by introducing new skills.

After a couple of years, when the hardware improved and multiuser systems made different forms of application possible, other research questions were considered. The place in which research on learning and teaching was done, was moved from the laboratory to the school. The influence of teacher strategies, pupil characteristics and classroom organisation on the use of the computer was investigated in the normal classroom situation. The "responsive environment" created by the computer was the basis for extensive field study in twelve primary schools in Amsterdam. The experimenters no longer controlled the situation, but restricted themselves to observations and to development of software upon request of the teachers and with their co-operation.

At the same time some classroom experiments were carried out in the field of problem solving: on learning a programming language and on information retrieval in ill-structured domains. The central question was, in this case, how the use of a computer could provoke real problem solving behaviour. The cognitive skills involved are related to the development of the integration of informatics in society, and are not traditionally dealt with in the classroom.

Adaptation to individual differences has been a central topic in all research during the last fifteen years. The traditional classroom organisation is in conflict with the fact that pupils differ widely on a number of variables related to the learning process. This problem may be attacked in two ways:

(a) adaptation of the pupil in the direction of the "average" student, to whom the education is leveled
(b) tuning the learning process to individual differences.

Both solutions complement each other. The educational situation itself defines the feasibility of both possibilities. The extent to which student-characteristics can be influenced varies tremendously. Van Muylwijk, Van der Veer and Waern (1983) suggest a dimension of changeability and mention four categories of learner variables (see figure 1).

stable, resistant to change ⟵			⟶ mainly defined by current influence from the outside world
personality factors	cognitive styles	strategies	personal knowledge structure
Intelligence	Field (in)dependence	Heuristic/ systematic	Production rules
Extraversion/ introversion	Visual/verbal		Schemata
Fear of failure	Operation/ comprehension learning	Serialist/ holist	Semantic network
Creativity(?)			

Figure 1 Dimension of changeability.

Personality factors like intelligence, creativity or negative fear of failure are conceived as stable and resistant to change. Knowledge and skills(production rules, schemes, semantic nets) result during the learning process: both content and internal representation are largely the result of the way of presentation. Between these two extremes are the domain of cognitive styles such as field-dependence, problem solving styles elaborated by Pask(1976), and the domain of strategies such as heuristic or algorithmic approach (De Leeuw,1983), serialism or holism. Strategies are conceived as domain-specific and adjustable, cognitive styles as rather stable products of talent and education.

Adaptation of the learning environment is more obvious if stable personality traits define the learning process. In the traditional educational situation, organised in levels of ability, only intelligence has been treated in this way. We have tried to apply the results of broader research in cognitive psychology within our experiments.

In the restricted scope of this contribution only a brief overview of the results will be presented. The experiments showed the relevance of internal and external conditions for the use of computers in the learning situation (section 2). The experience with the computer in the classroom is described in section 3.

This report is not a final statement. Technology is developing rapidly. Microcomputers are invading the schools and even the homes. Large amounts of money are being invested in the development of the fifth generation computers, with their new architecture and using the new concepts of human-computer interaction. The conclusions in section 4 are therefore only valid for the near future. Our results support those critics who believe in the irreplaceability of the teacher because of his

unique human faculties to react to a wide variety of educational problems in interaction with considerable individual differences. The computer is only a tool, albeit one with unique possibilities to adjust to individual learning behaviour. Technically speaking, this tool will certainly improve, but only a teacher who is able to work with this tool will be able to use it to its full extent.

2. INTERNAL AND EXTERNAL CONDITIONS FOR COMPUTER ASSISTED LEARNING

2.1. Knowledge and skills

In our study with primary school children the observation of Suppes(1979) was confirmed, that the opportunity to practise certain elementary skills turned out to be the most important aspect of the computer in this field of education. Our observations revealed that parallel to the educational evolution within which more attention is being paid to individual differences, training skills and becoming acquainted with a domain, or building-up of a personal knowledge base was the most favoured way of use. In this respect the computer was enlisted both for remedial purposes and for exercises after the introduction of new learning material.

A lot of training programs were written because teachers had requested them, especially in the fields of arithmetic, spelling and other language-aspects. For certain arithmetic exercises a considerable gain in learning was observed when the computer was used, in comparison to paper and pencil forms of the same exercises. These results were consistent and extended over periods of 4 months (Kok, 1984). The immediate feedback and the levels of help that our program offered, from a first general hint to increasingly more specific guidance if the error continued to be made, showed the unique possibilities of the computer in this field. Training children to understand texts led to other results. Providing feedback at increasing levels of specificity did not result in better performance compared to the provision of feedback one day after written exercises were done. Both methods resulted in a comparable amount of gain for pupils with reading difficulties. Their problems (not the identification of words but the interpretation of text in relation to knowledge that was already present) could to a certain extent be solved via simple delayed feedback. On the other hand we became aware of the problem of the interaction between the program and the student in "natural language", especially in a domain in which the objects themselves are fragments of natural language. Possible solutions are multiple choice questions, as in the actual training program, or open end questions for which the responses are matched against an internal list of alternatives. This last method should enable the student to be more "natural" in his responses and especially to be able to communicate more freely about the problem domain, but this would on the other hand make high demands upon the author of the computer program, and eventually ask for a solution in the sense of an "expert system" with an implicit model of the user. In our study the teacher could not be equaled in his precise diagnosis of individual misunderstanding.

In another study (Bernaert, 1978) we observed that students who worked in interaction with a teacher asked for more information and examples than their colleagues who were working with a computer program that provided these possibilities. From our observations we have concluded that the most preferable allocation of tasks could be to leave the introduction of new material to the teacher and to use the computer for exercises, in which students gradually need fewer examples, information and help. If however the computer is used for introducing new subject matter, the tutorial method is to be preferred. Only after the students have had enough instruction and examples should they be allowed to take control of the process, choosing the way they wish to proceed.

2.2. Strategies of learning

(a) Serialism/holism

These strategies are derived from the work by Pask (1976). They turn out to be dependent upon the specific task, like Pask suggested. If one matches the teaching strategy (the order in which the computer program presents the material) to the individual learning strategy a considerable gain in learning results is observed(Van der Veer, in the press).

(b) Search strategies

In tasks concerning the search of information from large databases two strategies may be distinguished, the one characterised by the immediate use of subsets of the data that are isolated from the whole, the other by an approach in phases, combining several subsets into one ultimate goal set, before actually screening the information in the resulting set. In domains unknown to the student, with an apparently vague structure, we found that the process of search can be facilitated by coaching the searcher to be consistent in his preferred method, although generally speaking the two approaches are equally successful (Beishuizen, 1984; Beishuizen and Van der Veer, in the press), as is the case for serialism or holism.

(c) Coding strategies

In learning to write computer programs, a distinction can be made between learning to structure one's problem solution (programming) and learning to code the solution into a computer language. The coding behaviour of novices differed systematically, but did prove to be related to the amount of experience in mathematics. University students with no mathematical background used abbreviations and one-letter identifiers whenever possible, deteriorating the readability of their programs, and giving them the idea that they had only learned tricks. Students with some years of mathematical education wrote programs which were understandable and they mentioned having learned a useful way of attacking problems from the programming course (Van der Veer, 1983). We found the time needed for mastering the coding language to be positively related to the amount of mathematical background, although students with or without a considerable amount of mathematical experience did not differ in performance on programming tests after the course.

Experience with a simple computer language designed in view of these results showed that primary-school children are able to master the skill of coding, using names that are workable to them and help them to remember the meaning of identifiers and procedures, provided their creativity does not bring them to choosing names with which the semantics associated are incompatible to the meaning in the program. In a few cases they even show progressive refinement in constructing the algorithm (Beishuizen and Van der Veer, in the press).

(d) Risk taking behaviour

The introduction of the computer as a new element in the learning situation may induce new learning strategies. When given a choice between doing exercises, examples or instruction, students showed a tendency to take more risks by switching to exercises earlier than they would have done in the normal situation, but because of this more time is spent practicing each unit. Especially the less-talented pupils showed this behaviour. Their results and scores on transfer tests are lower than in a setting in which the computer program prevents them from making their own choices, using an educational decision model instead (Bernaert, 1978).

(e) Heuristics and algorithms

Tutorial programs on problem solving either emphasise a more heuristic approach or teach an algorithmic approach to certain problems. This last strategy always leads to correct solutions within the problem domain, but this gain is of restricted value. Transfer to problems outside the primary scope is facilitated by using a heuristic approach, and in fact this method shows better results after a time lag of some days to a few months, even on the original problem domain (De Leeuw, 1983).

2.3. Cognitive styles

(a) Versatility

The cognitive styles that are developed from the conversation theory of Pask (1976) were used extensively in our project. These learning style variables have proven to be successful as predictors of learning behaviour in human-computer situations. Versatility (a flexible style in which a student may choose between concentrating on a general overview or focusing on individual operations) considerably facilitates the learning of a first programming language. Another factor in this model of styles, the tendency to learn (the amount of effort invested in collecting information), was related to the performance test which was given after a course in Cobol, predicting higher scores even if general intelligence was partialed out (Van der Veer and Van de Wolde, 1984). Once a computer language is mastered, these individual differences seem to be less important, but when problems of an obscure abstract nature have to be solved, non-versatiles turn out to be disabled, a disability that disappears as soon as an identical problem is stated in meaningful terms (Van der Veer, 1982).

(b) Field dependence

The tendency to actively structure the situation (field independence) without being affected by irrelevant aspects in the environment (field dependence) is an important determinant of problem solving behaviour. Field independent pupils are able to utilise a learned method or strategy (be it heuristic or algorithmic) in new, related domains. They perceive analogies and differences between old and new problem types, and they are more accurate in drawing schemes as an aid in problem solving (De Leeuw, 1983). Field dependent pupils on the other hand, need more assistance during the problem solving process, with levels of feedback varying from very general remarks to problem specific hints. Facilities like this are difficult to provide in the traditional school environments, but they can be provided in computer assisted situations. It is a feasible way to impose additional structure on the environment in which the students work.

2.4. Personality factors

(a) Intelligence

Adaptation of the content and structure of the learning material to the level of the pupil is the only possible way weaker students can be given a fair chance in education. Children with low scores on verbal and technical ability tests and on tests for educational achievement were not capable of choosing an optimal sequence of tasks within a learning environment. When the computer presented tasks according to a didactic principle they spent more time practicing, scored higher on a criterion test and they scored even better on transfer tests for related new learning domains. When working without the computer's "guidance", they risked too much by trying to do exercises at once, bypassing a sufficient amount of instruction and without looking at the examples (Bernaert, 1978). But even when coached adequately, the weaker pupils need more practice than do the gifted. There are computer assisted learning situations in which intelligence has no influence on the results, like learning to search in a database (Beishuizen, 1984) with the help of a computer coach. When learning the position value of numbers with the help of computer exercises, we found no relation between Raven-scores and the criterion test, although this relation is present with paper and pencil exercises. In this case the weaker pupils showed considerable gain from practice with the computer, but hardly any gain from written material (Kok, 1984). But these results cannot be generalised. In learning a programming languages, there is a significant correlation between criterion score and intelligence (Beishuizen and Brazier, 1984), consistent with reports on the Brookline LOGO project (Watt, 1979). Both in our project and the LOGO project, however, even the weak students were capable of showing a certain level of creative behaviour using a programming language, showing hidden talents that remained unexploited in traditional classroom situations.

(b) Negative fear of failure

In everyday school-life this handicap is responsible for much disappointment, frustration and, in the end, failure. From the start of our project a lot of attention has been paid to this problem, especially to the question of how fear of failure can be compensated, and how, using a systematic approach, it may even be cured (De Leeuw, 1983). Extensively structuring the situation is one form of compensation. Teaching an algorithmic approach, a method in which a solution is found step by step, combined with the security of a valid solution, helps to overcome the fear of failure, even in related problem domains, but only if the original domain was not too narrow. Students without this handicap profit more from a heuristic approach. After a considerable time delay, in fact, a heuristic method gives the most stable results for all pupils. We are not too happy with this conclusion: We can only compensate negative fear of failure for closely related domains, for a short period of time. Another observation in the same experiment is more profitable: students with negative fear of failure needed more help to correct errors. If the computer program first presented feedback consisting of very general remarks about the problem being considered, presenting more specific information at a later stage, the need for help decreased. In this experimental situation we could say we had observed something which could be referred to as a cure rather than a compensation. Another positive effect of learning with computers is the lack of competition, so often perceptible in normal classroom situations. The computer is a non-human object, and pupils experience this as a safe situation, even when they are in fact keeping pace with other students.

3. THE COMPUTER IN THE PRIMARY SCHOOL - A RESPONSIVE ENVIRONMENT

To investigate the effect of computers in the school, we explored the situation in the highest grades of 12 normal Dutch primary schools. The age of the pupils in this population is about 10 - 12. In this environment the educational goals are changing - more differentiation is being strived for within the class and more attention is being paid to individual differences. These differences should be observed to their full extent at primary school level, because the children are sent to different secondary schools depending on their capabilities. Experience in primary schools shows first of all that the introduction of new material can well be left to the teacher. The computer is used for independent practice in a variety of problem solving domains. It does not replace the teacher, but it is a device with special qualities, a possibility to enrich the learning environment. In evaluating its possibilities the researchers refrained from intervention in the teaching method, apart from the introduction of a pupils programming language and a couple of field experiments.

3.1. The "study machine"

The integrated system available enabled teachers to independently decide when, for which pupils and for which task the computer was used. The "computer" as we provided it for the school, consisted of one terminal, connected via a dial up telephone line to a system that was very easy to use (e.g. the Unix system was masked). The time the study machine was in operation in the classroom varied between 1.5 and 16 hours a week, a student-session took between 10 and 30 minutes. The best place for the terminal turned out to be in the classroom. The teachers gave ideas for new lessons, helped to define didactic aims and created exercises themselves, especially in the domain of arithmetic, language and geography. In this way the study machine developed into an environment within which simple exercises, structured learning units, problems in the form of games, tests and a programming language are the elements. Some teachers used the study machine for group-education, others for individuals who needed special attention or for remedial aims.

(a) Arithmetic
Exercises in this domain are problem oriented: After an error the problem is gradually divided into smaller units. The exercises are grouped into levels of uniform difficulty and after every 10 items an overview of the results is given. In some programs a recommendation is given about the best level to continue practise. The domain includes simple operations like addition, subtraction and division. For multiplication, proportions and percentages remedial use was observed to be very successful. Exercises about linear measures were often combined with a lesson about their application. For exercises on the manipulation of digits in numbers a field-experiment showed a considerable learning result, that even after 5 months was still superior to the result of written exercises.

(b) Language
Three different template lessons are available in this domain. The teacher can insert new material in a very easy way. Especially spelling exercises have been very popular, mostly for remedial goals but sometimes in group instruction. A considerable gain in learning has been observed. Together the teachers created a total of about 150 different spelling lessons, but some teachers only used the exercises their colleagues constructed. Some teachers even insert new lessons which can only be used by a few pupils with special spelling problems.
A second template lesson provides the opportunity to create lessons in which a pupil has to fill in the appropriate words, which can be chosen from a set of words provided along with the exercise. In this way another 150 lessons were written by teachers, about grammer and proverbs, Turkish-Dutch translation of phrases, but also lessons about arithmetic problems, geography and history.
These types of template lessons were used quite frequently. More time was spent using these two template facilities (with their many available exercises) than on any other part of the study machine. Textual explanation is the subject of the third template.

(c) Games
The games in the study machine are included because of specific problem solving behaviour required to win - systematic search within well-known domains such as numbers, restricted by properties like primes, squares or divisibility. A number of game lessons are constructed in levels that may be chosen according to the problem solving behaviour of the student, sometimes done automatically by the program, sometimes only in the way of an advise to choose another game.

(d) Tests
A special facility for administering educational tests is provided, enabling the student to determine not only his own performance, but also the amount of certainty and the reliability of his knowledge (Dirkzwager, 1975).

3.2. A programming language in the primary school

Based on previous experiments with programming language constructs and their relation to individual differences, a programming language was defined that combined neat control structures, readable naming, interactive editing and automatic syntax checking. A problem domain dependent standard prelude releases users from administrative bother and provides the solution to detailed problems that are not relevant to the pupil or the problem solving process. Learning the language with the help of a written guide took 6th grade pupils 30 hours. Case studies have shown that students are capable of creatively manipulating with problems, just like Watt (1979) found in the Brookline project. Coding is readily mastered at this age. Pupils are able to produce examples and applications of syntactical concepts. The creation of an original program of any volume however may be found with only a few gifted children. This is in accordance with a recent statement by Weizenbaum (Foppema, 1983): "A certain amount of maturity is a condition to create a real program. You will not expect a 14 year old to do the job of an engineer".

3.3. Opinions and attitudes in the school

(a) Pupils
For several years running, a questionnaire was filled in by the pupils, that showed a positive evaluation of the computer as such. If it was left to the students, they would spend between 2 hours a week to 2 hours a day (which they are in fact never allowed by their teachers) working with the computer. The student's positive attitude remains stable even once they are accustomed to the situation having worked with computer up to 3 years. The most important arguments they mentioned were: when making an error one is not laughed at; feedback is provided immediately and errors can be corrected right away; one can keep to ones own pace; typing is preferred to handwriting (writing difficulties are the cause of problems in a variety of other domains at school). During the years the arguments given to justify the preference for computer exercises shifted from the arguments based on the learning gain expected to such arguments as "the computer is pleasant AND instructive", and "one learns more in a pleasant situation".

(b) Teachers
Arguments in favour of the use of a computer concentrate on one hand on individualisation, an approach which focuses on the pupil's weak points, and on the other hand on difficult parts of the learning material. The actual results can never be fully credited to the computer. A computer is a tool, that may be useful in combination with other tools. Topics such as: improvement of spelling and reading, a variable approach to learning material, the practise of standard algorithms and the enhancement of motivation, are mentioned as positive effects. Teachers observed that children are not afraid to make errors and that they sustain the search for solutions longer than in other situations.

4. RECOMMENDATIONS AND CONCLUSIONS

Haylock(1983) posed three questions about the way in which the present education and the new technology interact:

(a) How the computer "can be used to do more effectively and efficiently what teachers are doing already by other means";
(b) "How they can be used to enhance the existing curriculum by making possible learning experiences not previously practicable";
(c) "The possibility of new components on the content of the primary school curriculum".

Our project has tried, at least partially, to answer these questions.

4.1. The role of the teacher

The creation of a tutorial program is a specialist's job. Experience with an author language or with a high level programming language is a prerequisite, as is either knowledge of the learning material or the cooperation of an expert. The ratio of 100 hours of development to one hour of tutorial program often stated is quite probable. In the primary schools there does not seem to be much need for this type of C.A.I. Teachers do not need be replaced. Venezky(1983) mentions the "teacher directed classroom", in which the computer is only an instructional tool, that facilitates individualisation. Our observations confirm this idea: practise of skills, problem solving exercises and the coaching of strategy development are tasks which can and in fact do use computer facilities. Some tutorial aspects are recognizable in these programs, aspects which are relatively easy implemented e.g. in arithmetic. The interaction in the case of language education asks for more advanced design techniques. "Simple" solutions like multiple choice are not effective.

A well chosen system does not require an intensive education of the teacher. Regular contact with colleagues using the same type of system is very important and leads to exchange of programs and methods.

4.2. Efficient applications

The most frequent task for which computers are employed in the primary school is practising cognitive skills. Alternation with traditional means provokes motivation, as does the direct feedback and the possibility to determine ones own pace. The possibility to refine the "help" in phases is efficient for less skillful pupils, e.g. in arithmetic.

This "help" should not be thrusted upon a student. They themselves perceive its usefulness "for certain mistakes". The best method is to give a pupil a second opportunity to recover the error before the help is offered. This enables the less talented to learn from the exercises.

The arrangement of the exercises in levels enables the student to transfer to a level of difficulty based on a diagnosis of his ability. Often the student will decide what to do next depending on previous results, in other cases the teacher determines the next level after evaluating the progress. A third possibility is to leave the decision to the program. Although this seems to be very adaptive (to the level of the student), it is not very transparent. The student might develop the idea the computer is more "intelligent" than he himself is, or he might get irritated, because a fixed decision rule dominates human reason.

Template lessons that may be filled in by the teacher, are naturally simple in structure and uniform in feedback. This appears to have been accepted: the pupils themselves stated that the direct feedback and the possibility to correct mistakes was the most important characteristic of these programs. We observed that the increase in learning confirms these statements.

4.3. Enrichment of the learning environment

In the traditional classroom situation only restricted possibilities are available for the adaptation of the teaching strategy to individual differences. The application of a computer enables us to develop methods based on cognitive psychological theories.

Learning processes with problem solving aspects are best conducted with heuristic strategies. Systematic assistance at the stage in which that strategy is to be discovered leads to long term gain and positive transfer to new problem domains. An algorithmic method is only momentarily efficient but does not help a general approach.

Individual differences between serialist and holist strategies, a task dependent variable, may be compensated by adapting the teaching strategy of the program to the student. When searching for information in a data base, for instance, the program may determine the personal strategy of the pupil and coach him, if his behaviour deviates, by advising what step should be taken next, thus improving the results.

Negative fear of failure may be compensated by providing help facilities that are structured from general to specific. Gradually the pupil will develop independence in the task domain. Field dependent pupils also profit from being given the possibility to structure the situation.

Independence in applying for help and in determining the order of exercises is a useful aspect in the learning environment. But less talented children lack the ability to know when and how much information they need. In that case a computer program structured according to didactic principles is successful in guiding the learning process whereas the more gifted children profit more from freedom.

4.4. New elements in the curriculum

(a) Learning a programming language
A coding language structures the problem solving process and at the same time enables creative thinking, provided the student is allowed to work on his own level. A number of conditions should be fulfilled.
- the programming exercises have to be meaningful: the semantics of the problem should correspond to the foreknowledge of the student;
- a standard prelude prevents administrative difficulties; most teachers rely on a specialist to implement the domain specific elements for this prelude;
- The language will provide meaningful basic symbols, though pseudo-natural language has to be avoided; assignments and initialisation should be conceptually comprehensible to the novice; control structures should correspond to the naive way of thinking of the student.

Languages like BASIC do not fulfil these demands.

(b) Searching for information
The computer can also provide facilities for students to become familiar in new, for the student ill-structured domains: The skill of learning to learn will increase in importance in the education of the future. An instructional database may be constructed with an efficient keyword structure. The importance of taking notes should be stressed by the teacher. This optimises the results and prevents passivity. The structure in the database must provide opportunities both for comprehension learners and for operation learners, to work according their individual style.

(c) Games
Cognitive skills like structuring a set of elements and developing general rules may be stimulated by well constructed computer games, in which the level of difficulty may be either chosen by the student, with the help of an advise by the program, or determined automatically, in order to optimise motivation and effort.

(d) Self testing
The possibility to determine ones own level of knowledge and insight, and the reliability of the results without having to depend on the teacher, promotes the metacognition of the student.

REFERENCES

Beishuizen, J.J. (1984). Informatie verzamelen in een bibliotheek: coaching en zoekstrategieën. Leren met computers in het onderwijs. Dirkzwager, A., Fokkema, S.D., Veer, G.C. van der, Beishuizen, J.J. (Eds.). S.V.O., Den Haag.

Beishuizen, J.J., Brazier, F.M.T. (1984). Leren programmeren op de basisschool. Leren met computers in het onderwijs. Dirkzwager, A., Fokkema, S.D., Veer, G.C. van der, Beishuizen, J.J. (Eds.). S.V.O., Den Haag.

Beishuizen, J.J., Veer, G.C. van der. (in the press). Een responsieve leeromgeving voor hoogbegaafde kinderen. Hoogbegaafden in onze samenleving. Mönks, F.J., Span, P. (Eds.).

Bernaert, G.F. (1978). Sturing in het onderwijsleerproces, cognitieve capaciteit en leersituatie. S.V.O., Den Haag.

Dirkzwager, A. (1975). Computer-based testing with automatic scoring based on subjective probabilities. Computers in Education. Lecarme, O., Lewis, R. (Eds.). North-Holland, Amsterdam.

Foppema, R. (1983). Huiscomputers zijn niet nuttig. Trouw, 28 december 1983, 12.
Haylock, D. (1983). Computers and Children in the Primary School. Journal of curriculumstudies, 15, 230-231.
Kok, E.J. (1984). Effectiviteit van computeronderwijs - getalstructuur. Leren met computers in het onderwijs. Dirkzwager, A., Fokkema, S.D., Veer, G.C., Beishuizen, J.J. (Eds.). S.V.O., Den Haag.
Leeuw, L. de (1983). Teaching problemsolving: an ATI study of the effects of teaching algorithmic and heuristic solution methods. Instructional science, 12, 1-48.
Muylwijk, B. van, Veer, G.C. van der, Waern, Y. (1983). Behaviour and information technology, 2, 313-326.
Pask, G. (1976). Styles and strategies of learning. British Journal of educational Psychology, 46, 128-148.
Suppes, P. (1979). Current trends in Computer-Assisted Instruction. Advances in Computers. Rubinoff, M., Yovits, M.C. (Eds.). Academic Press, New York.
Veer, G.C. van der (1983). Individual differences in cognitive style and educational background and their effect upon the learning of a programming language. Psychologie des Programmierens. Schauer, H., Tauber, M. (Eds.). Oldenbourg, Wien.
Veer, G.C. van der (in the press). Learning style in conversation - a practical application to man-machine interaction. Cybernetics.
Veer, G.C. van der, Wolde, J. van de (1982). Psychological aspects of problem solving with the help of computer languages. Computers and Education, 6, 229-234.
Veer, G.C. van der, Wolde, J. van de (1984). Leerstijlen bij mens-machine interactie - een bewerking van de smokkelaarstest van Gordon Pask. Leren met computers in het onderwijs. Dirkzwager, A., Fokkema, S.D., Veer, G.C. van der, Beishuizen, J.J. (Eds.). S.V.O., Den Haag.
Venezky, R.L. (1983). Evaluating Computer-Assisted Instruction on Its Own Terms. Classroom Computers and Cognitive Science. Wilkinson, A.C. (Ed.). Academic Press, New York.
Watt, D. (1979). Final report of the Brookline LOGO project, part III: Profiles of individual student's work. M.I.T., Cambridge, Massachusetts.

INTERFACES IN THE FIELD

A REALISATION OF A HUMAN-COMPUTER INTERFACE FOR NAIVE USERS - A CASE STUDY

Günter Haring, Theodor Krasser

Technische Universität, Graz
Austria

The realisation of a human-computer interface for an information storage and retrieval system used by the staff of a company in mechanical engineering industry is described in this paper. The system had to be designed according to the needs, skills and data processing background of this user group, taking the tasks to be performed into consideration. The system design process, based on human factor design goals and integrating quality control, is compared with the usual software development procedure. The description of the system explains the way in which different dialogue tools such as menu selection, form filling, function keys etc. have been integrated. Data entry and query functions are used as examples.

1. INTRODUCTION

The staff of a company with activities in design and development of internal combustion engines was faced with a rapidly increasing amount of information on engines (mostly test results), which needed processing. An information storage and retrieval system in which the results of previous developments were stored, was developed to reduce the cost of improvements and development and to specify requirements at the beginning of each new project. For the realisation of the system the programming language FORTRAN 77, a forms management system, and an interactive query and reports system with a call interface for FORTRAN-programs, were available. Nearly all members of the staff, who were to use the system, had little or no knowledge of data processing (naive users). They wanted to be able to use the information storage and retrieval system as a tool for solving their problems without having to invest much time and effort. The theoretical justification of a system as such (in which the user's skills and requirements were taken into consideration) will now follow.

2. DESIGN GOALS

The aim and purpose of a design process is to develop a software-technical solution to the problems which arise when the user's requirements are to be translated into concrete facilities within the software product (Balzert, 1982). Because the types of users and the tasks they wish to perform differ from system to system, system designers must have a thorough understanding of the needs and skills of the users and the tasks that must be accomplished (Shneiderman, 1983). In principle we can divide the design goals into two categories: primary design goals and human factors design goals.

2.1. Primary design goals

The primary design goals are
- to ascertain the correct functioning;
- to ensure reliability;
- to be on schedule and within the budget (Shneiderman, 1983).

(a) Functioning
Most of the system activities occur at a level unknown to the users. The user's knowledge of the system is limited to the set of functions visible at the interface level. The system designers must analyse the user's image of the task before correct functioning can be guaranteed. Tasks frequently executed are easy to detect, but it is often a problem to discover occasional and exceptional tasks. One has to avoid not only inadequate functionality but also confusing excessive functionality (Shneiderman, 1983).

(b) Reliability
The software architecture and hardware support must guarantee, that the system is accessible at any time when the user needs it. Furthermore maintenance must be easy and performance should be correct. The user must be informed on all effects of his input to the system and the system must have a chance to continue without problems. Important questions concerning data protection, security and information integrity are also related to reliability (Shneiderman, 1983; Dehning et al., 1982; Date, 1976).

(c) Observance of fixed dates and budgetary plans
The third primary design goal is to be on schedule and within the budget. Exceeding terms and costs may, as Shneiderman points out, enrage the potential users and may result in their total rejection of the system.

2.2. Human factors design goal

The primary design goals having been described, our attention can be focussed on the design goals concerning human factors. They include:
- a user interface, that is adapted to the user group with on-line help and function keys;
- ease of use and ease of learning;
- user friendly error handling;
- response time requirements;
- data representation.

There are examples of clever designs for communities of users which may be inappropriate for other communities. This implies the importance of an adequate interface for specific user group.

2.3. Man-machine interface

When designing the interface the dialogue style (natural language based, command language, form filling, menu selection), function keys, on-line help, (error)messages and display of messages and output of data should be considered. This chapter describes the theoretical background of the man-machine interface characteristics, which were relevant to the system we have developed.

2.3.1. Menu selection and form filling

The user has to select one of the possibilities the system offers (menu selection) or has to fill in the blank spaces of skeletons with the required data (form filling). These types of interaction are mainly machine initiated. The advantages of these types of interaction are:

- the user is guided by the system, the time to learn is short;
- it is easy to recognise which functions are available;
- it is obvious which values are required by the system;
- syntax errors are avoided;
- if there is a mask and menu management system, program code is drastically reduced (Fischer and Zeidler, 1982; Quiniou and Saint-Dizier, 1982).

On the other hand there are some disadvantages:
- it is difficult to propose items which semantically correspond to those intended by the user;
- the dialogue is static and sometimes very long;
- there are only a limited number of responses available;
- complex menus may confuse the user;
- there is no possibility to combine commands (Fischer and Zeidler, 1982; Quiniou and Saint-Dizier, 1982).

The following points should also be taken in consideration with form filling and menu selection:
- if the structure of the problem is well understood, the menu structure should represent this;
- the direct choice of frequently used functions should be possible;
- use a menu hierarchy to structure a large number of choices;
- it must be possible to go back to the previous menu if the selected menu was selected by mistake;
- as a rule one menu contains five to nine choices; but don't split up natural groups (e.g. twelve months);
- the last entry selected should be on the first line of the new menu;
- it must be easy to distinguish between the different entities;
- it must be easy to move from field to field in forms;
- the forms indicate the expected format for the values;
- menus and forms should be independent from program code to allow an easy adaptation to new requirements (Brown, 1982; Nagler, 1982; Miller, 1956; Schmidt, 1983).

2.3.2. Function keys

The user has the possibility to select frequently used functions by typing a key on the keyboard. Examples are cursor manipulation, editing functions and the HELP-function. The advantage of function keys is that they reduce the amount of time spent typing. On the other hand the user has to keep in mind the position of the keys and their functions. Only a small number of function keys should be used to avoid the user having to remember too many keys at once and thus wasting time searching for appropriate keys. He must be able to obtain a list of the correspondence between functions and keys. The function keys should be software controlled, so that this correspondence can be reprogrammed when required. The user should be able to define his own function keys. It should also be possible to perform a function by entering a command name instead of using the function key.

2.3.3. Help functions

Help functions provide on-line help in case of difficulties. These may be asked for by the user or be given automatically by the system (Dehning et al.,1982).

2.3.4. (Error)messages

After the operator has fed input into the system, the system should always respond so that the user knows that his input has really travelled all the way to the computer (Martin, 1973). Very important are messages which indicate an error. The system designers should make sure that error messages have a positive emotional tone. The error message should not make the user feel guilty. It seems very

important that the message should describe the error in detail, so that the user is able to recognise the reason for the error. The system may also inform the user how to continue (Ledgard et al., 1981).

2.3.5. Display of messages and output of data

In principle output can be done in speech form, in text form or in graphic form. Output in speech form can only be used in a limited number of cases, usually with fairly restricted possibilities for the user (Nes, 1982). The most commonly used output medium is the video screen. It is used for the display of system messages and for output of data. It is important for the user to discern the different outputs (messages, results, help information). This can be realised either by using different letter types (upper case, lower case, different distances), or by partitioning the screen into different regions, or by using the video functions of the screen. For the representation of results the system designer should consider the following aspects:

- the output should be self-descriptive;
- the vocabulary should be consistent with natural language, using common abbreviations and with input vocabulary;
- avoid words and special characters which may not be known to the user;
- avoid irrelevant information;
- choose a problem specific representation of the information, using as few objects as possible;
- find a clear and structured arrangement of information. It must be possible to recognise the most important parts immediately;
- erase information from the screen when it is no longer needed.

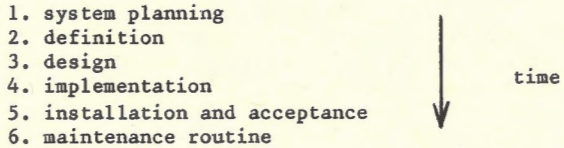
For large amounts of information the system designer must also consider the following:

- the user has the possibility of selecting the data he wants to see;
- there are facilities to interrupt, to continue and to cancel the output;
- the user decides when he wants to see the next unit of information;
- use scrolling techniques and paging;
- allow the user to select the output device. Often it is desirable to have an output listing on paper;
- if possible, use graphical output facilities.

In this chapter we have defined some design goals and characteristics of a man machine interface for naive users. In the next chapter we will be interested in the question of how these design goals can be achieved.

3. SYSTEM DEVELOPMENT

The system designer must develop the system taking the needs, skills and data processing background of the user group and the tasks they want to perform, into account, to achieve a satisfying solution for the software system. We will use the following phase division of the software life cycle (Balzert, 1982) to compare the usual software development procedure with a system design process which integrates quality control and a model of the human user:



3.1. "Common" process of system development

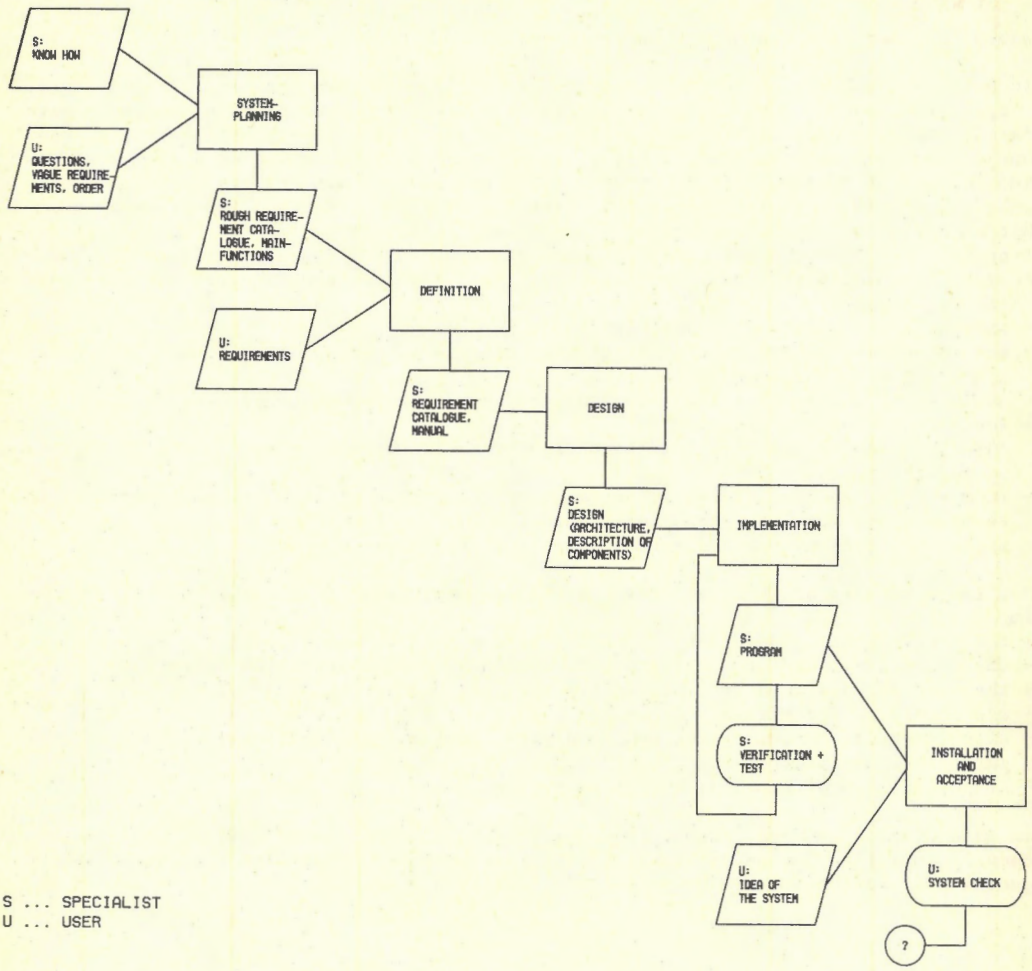


Figure 1 "Common" process of software-system development.

The common process of system development and the role of the user is shown in figure 1. Persons, who want to solve a problem with computers consult DP-specialists. They cooperate during the system planning phase, check the feasibility, define a rough list of requirements and the main functions based on the - often vague - ideas of the potential users. In the design phase these documents and an exact analysis of the requirements lead to a complete list of requirements and a manual with the functional description of the system. Later the requirements are transformed into a software-technical design, which is implemented and tested during the implementation phase. In the installation and acceptance phase the users (and their ideas of the system) are confronted with the real system. In this process the linear flow and the limited communication with the user are disadvantageous. Often system designers

have difficulty cooperating with users whose ideas are rather vaguely defined. This lack of communication and the late quality control by the user at the end of the software development cause the late and therefore expensive detection of errors and misunderstandings.

3.2. System development integrating a model of the user and quality control

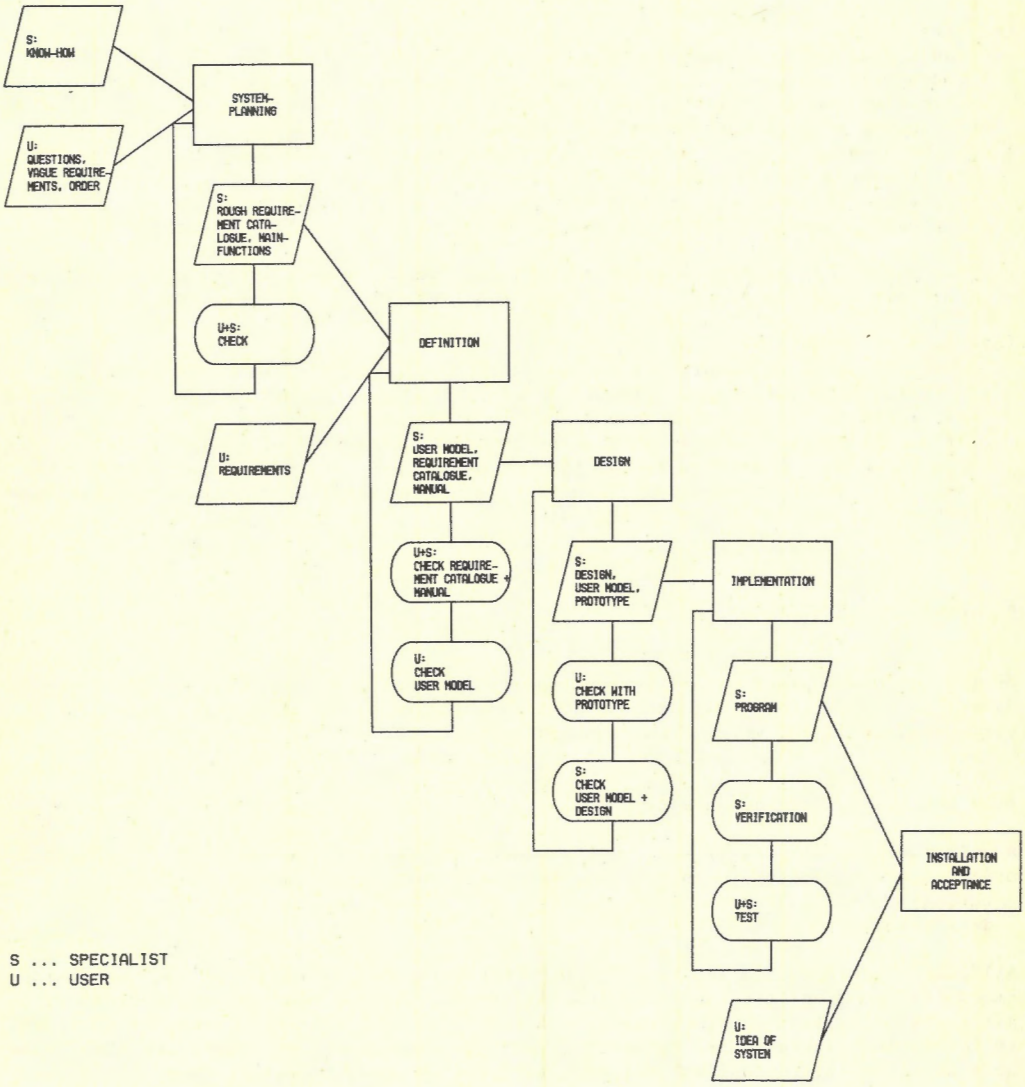


Figure 2 System development as an iterative process with integrated quality control and integration of the users.

Apart from this disadvantage and the aim to achieve the design goals there are also organisational changes, that are necessary for system development and installation, which requires the integration of the users in the system development process.

Figure 2 shows the system development procedure integrating quality control and a model of the user. In contrast to the "common" process, the results are checked after each phase. If the results don't stand the test the phase will be run through once again. The check is done by both the specialist and the user. In addition to the known methods for getting to know the user's wishes, such as interviews, questionnaires and analysis of these parts of the process that have defined, two methods should be mentioned:
- user model,
- preliminary prototype models.

(a) User model
The system designer develops a model of the user, that is based on past experiences with the user. With this model they consider those fields of planning and execution, which should really be controlled by the user (Dzida, 1982). The following aspects have to be considered when constructing a consistent model:
- the needs and skills of the user and factors, that influence their behaviour;
- the user's view of the problem;
- peculiarities of the problem;
- possibilities for designing the man machine interface (Nagler, 1980).

The model only considers those aspects of input/output behaviour, which are relevant for the user, and those functions which the user knows of (Nagler, 1982).

(b) Rapid prototyping
The list of requirements and the manual supply insufficient information about working with the system. It is therefore important to make a "complete" system available to the users to enable them to elaborate and correct their view, by using the system. This can be done by creating rapidly prototype models, which can be seen as a part of the design process. Only the functional characteristics of the prototype are important, whereas inefficient computer performance and memory allocation are insignificant.

4. THE ACTUAL SYSTEM

During the development of our information storage and retrieval system for engine data, the model of the user as well as quality control during all phases of the software life cycle, were taken into consideration.

4.1. Description of the data structure

A relational data model of the engine data was the basis for the application oriented data structure. In principle it distinguishes two structures:
- tables;
- data files.

All characteristic data for the internal combustion engine (that may be of importance for queries) are represented as values in tables. Further information, especially test results, is stored in data files. For each data file there is an entry in a table linking the data file to a specified engine. There are also links between different tables, but the user need not be aware of these links.

4.2. Functional description

This structure and the tasks that the user wants to perform required the following functions:
- functions for data storage and update in tables;
- functions for information retrieval;
- functions that allow the manipulation of data files.

In detail this can be split up into eight main functions:
- insert a row in a table;
- modify a row in a table;
- delete a row in a table (and all the corresponding links);
- define and execute a query;
- execute a predefined query;
- get and convert a data file (to use the data for further computations, to plot diagrams or to print the results);
- put a data file;
- delete a data file.

4.3. Interface description

In our application the most appropriate dialogue style for an interface seemed to be menu selection and form filling, because a machine initiated interaction satisfies the needs of nearly all the users, who prefer to be guided by the system, want to minimise the learning time and expect the system to prohibit them making errors (see 1 and 2.3).

After starting the system a menu with the eight main functions is shown to the user. By typing the number corresponding to the function the user selects the option. All other functions and forms use the right upper corner of the screen to display the function last selected. Every field shows the maximal length of the input value and every input is checked immediately if possible. If the execution of an action is to last longer than a few seconds the user receives the message "system working".

For the function "INSERT A ROW IN A TABLE" the user must specify a table and he may also specify a row identification. If he specifies a row identification the new record is initialised with these values. The user has to fill (or modify if he has initialised the record) the fields of the forms displayed.

For the function "MODIFY A ROW IN A TABLE" the user must specify both the table and the row identification. The corresponding values are shown in forms and the user can update the distinct field values.

For the function "DELETE A ROW IN A TABLE" the user must again specify table and row identification. The specified record is shown to the user and he must verify wanting to delete the record.

Once a table has been specified it is used as the default value for further actions. The user can use the function keys of the keyboard for manipulating data in tables and for other functions. The following functions are available:
- move cursor one position to the right in the current field;
- move cursor one position to the left in the current field;
- move cursor to next field;
- move cursor to previous field;
- move cursor to first field of form;
- move cursor to last field of form;
- previous form;
- next form;
- first form of a logical record distributed over some forms;

- last form of a logical record distributed over some forms;
- delete field value;
- undelete field value;
- refresh screen;
- help to get more information about the current input field and the meaning of the function keys;
- command to interrupt the current function.

The function keys are software controlled, but the user is not able to define his own keys.

For the definition of a query the user gets a form and he has to enter:
- mode (batch or interactive);
- table(s), that contain values that shall be compared or printed out;
- boolean expression;
- name of the fields (attributes) for which the user wants to see the values (the names of the field are the same as in the forms for data manipulation, for every field there exists also a short name);
- output unit (terminal, file, line printer).

To define a boolean expression the user must specify:

field name, relational operator, argument list
[,boolean operator (and/or), field name, relational operator, argument list]

The square brackets indicate, that the expression may occur zero or more times. If the first field name is blank, the whole table is selected.

When the user selects the function "EXECUTE A PREDEFINED QUERY" a menu with these queries is shown to him. The system prompts for the necessary parameters depending on his choice.

For the file handling functions the user has to enter an input file name. For "GET FILE" and "PUT FILE" he can also specify an output file name, otherwise a default file name is used.

5. CONCLUSIONS

We have described the realisation of a man machine interface of an information storage and retrieval system, (mainly) used by naive users. We have presented a number of design goals and have focussed our attention on the requirements for a man machine interface for naive users. Furthermore we have given an overview of the system design procedure, in which the model of the user and quality control are integrated into all phases of the software life cycle and we have compared this procedure to the common process of system design. Finally we described the main characteristics of our system.

REFERENCES

Balzert, H. (1982). Die Entwicklung von Software-Systemen (Prinzipien, Methoden, Sprachen, Werkzeuge). Reihe Informatik 34, Bibliographisches Institut, Mannheim.

Brown, J.W. (1982). Controlling the Complexity of Menu Networks. Communications of the ACM, 25, 412-418.

Date, C.J. (1976). An Introduction to Database Systems. Addison-Wesley, Amsterdam.

Dehning, W., Essig, H., Maass, S. (1982). The Adaptation of Virtual Man-Computer Interfaces to User Requirements in Dialog. Springer Verlag, Berlin.

Dzida, W. (1982). Dialogfähige Werkzeuge und arbeitsgerechte Dialogformen. Informatik und Psychologie. Schauer, H., Tauber, M.J. (Eds.). Oldenbourg, Wien.

Fischer, H.G., Zeidler, A. (1982). User Friendly Interaction with an Integrated Data Base Information Retrieval System. Zurich Seminar on Digital Communications (Man-Machine Interaction), IEEE Cat. no. 82CH1735-0, 153-158.

Ledgard, H., Singer, A., Whitside, J. (1981). Directions in Human Factors for Interactive Systems. Springer, Berlin.

Martin, J. (1973). Design of Man-Computer Dialogues. Prentice-Hall, Englewood Cliffs.

Miller, G. (1956). The Magical Number Seven, Plus or Minus Two: Some Limits of Capity for Processing Information. Psychology Review, 63, 81-97.

Nagler, R. (1980). Entwurf und Realisierung von Dialogsystemen. Dissertation, Wien.

Nagler, R. (1982). Entwurf benutzerfreundlicher Dialogsysteme aus der Sicht menschlicher Informationsverarbeitung. Informatik und Psychologie. Schauer, H., Tauber, M.J. (Eds.). Oldenbourg, Wien.

Nes, F.L.van (1982). Perceptive, Cognitive and Communication Aspects of Data Processing Equipment. Zurich Seminar on Digital Communication (Man-Machine Interaction), IEEE Cat. no. 82CH1735-0, 259-262.

Quiniou, R., Saint-Dizier, P. (1982). Man-Machine Interface for Large Public Applications. Zurich Seminar on Digital Communications (Man-Machine Interaction), IEEE Cat. no. 82CH1735-0, 147-152.

Schmidt, E. (1983). Computer sollen menschenfreundlich werden. Computerwoche 1983, 21, 20 Mai.

Shneiderman, B. (1983). Human Factors for Interactive Software. Enduser Systems and Their Human Factors. Blaser, A., Zoeppritz, M. (Eds.). Springer, Heidelberg.

REAL TIME GRAPHIC SIMULATION OF VISUAL EFFECTS OF EGOMOTION[1])

Patrick Peruch*, Viola Cavallo*, Christian Deutsch+, Jean Pailhous*

*Laboratoire de Psychologie de l'Apprentissage, Marseille
+Société Opeform, Malakoff
France

Both spatial displacements and their visual consequences which allow analysis and control of trajectories are now an important research topic. This activity plays a large role in our everyday life and so it largely exceeds the preoccupations of cognitive psychology: for example, it poses some problems for the ergonomist who has to elaborate graphic job-aids for manoeuvring large ships.

We present 3 important aspects of our method:

(a) A multi-disciplinary team (mathematics, software engineering, ergonomics, cognitive psychology) works on a collective project.
(b) We have constructed a model of the mental (cognitive) processes involved in this task.
(c) We have produced dynamic images in real time with a computer and graphic display; we have simulated some aspects of active movement in definite spaces.

1. INTRODUCTION

At present much applied research is concerned with visual control of displacement, and in particular with situations in which man is transported. In this type of situation vision is most important because it is the main modality for spatial data processing. Visual information caused by displacement includes information about an apparent motion of the surrounding space elements. Several authors (Gibson, 1950, 1979; Gordon, 1965; Lee, 1974; Nakayama and Loomis, 1974) have described these "optical flow properties", related to the trajectory. Bonnet (1975) described two systems: a "Motion Analysing System" (MAS), and a "Displacement Analysing System" (DAS). Both systems are applied to high velocity motion, whilst information on both acceleration and deceleration, and on modification of the trajectory are supplied by the vestibular system. Car driving is a good example of this situation. At low velocity, only positional information (supplied by the DAS) is available and can be used to discover the trajectory properties. In such a situation, maritime navigation is an example, the amount of data available is not sufficient for displacement analysis,making calculations from spatial positions necessary. Another example is the minute hand of a clock which seems motionless but even so changes its position.

We will now approach the question of large ships manoeuvring in harbour areas. In such places the difficulty of controlling the trajectory is mainly due to the narrowness of the channel, the streams, the inertia and the lack of manoeuvrability of the ship. The navigator needs dynamic data about the ship trajectory, which he cannot obtain from static job-aids (maps, nautical instructions, maritime buoyage systems). Some studies on work analysis (for example, Brouard et al., 1979) have shown that radar can be useful, but the indications which are used to control the trajectory are mostly deduced from the apparent transformations of the surrounding space. Each type of movement (translation, rotation and their combinations) induces

characteristic modifications of the visual scene. As a matter of fact, pilots in training are taught to use these indications efficiently: precise data on direction and speed can be given only by considering the sequence of the images. The use of these geometric optical transformations, these "spatial consequences" of displacement (Pailhous and Cavallo, 1982), is essential during night-navigation; most depth perceptual indications are not available: there are perturbations of shapes and apparent size perception, and no gradient of density of texture. The more the constraints increase (in situations such as supertanker piloting, night-manoeuvring, bad meteorological conditions, and so on), the better adapted job-aids are to facilitate, or even permit large ship navigation in harbour areas. We took part in a research program, the purpose of which was two-fold:

(a) to evaluate the interest in the use of radio-electrical aids;
(b) to develop new modes of presentation of data which would add to or even replace the usual visual aids.

This research program is now in the experimental stage.

2. RESEARCH PRESENTATION

2.1. The model

One way of studying human behaviour in work situations consists of constructing a model of mental (cognitive) processes and of simulating them within experimental situations. In our case, we were rapidly confronted with the use of computers and the need for a multidisciplinary team, with psychologists and mathematicians on the one hand and software engineers and ergonomists on the other. Collaboration of mathematicians and psychologists (specialised in spatial data processing study) facilitated the modelling of mental activities. Software engineers and ergonomists could then concretise the results of these fundamental studies, producing software to help pilots.

The experimental apparatus we used consisted of a Hewlett-Packard 9825 computer and a graphic display. The real time program showed a scene simulating some aspects of active driving in definite spaces. The simulated displacement took into account both the ship inertia, and the stream and approximations of the port radio-localisation system. The subject was seated in front of the screen and gave instructions about the modifications of direction and speed which were thereupon executed by the experimenter via his computer keyboard.

2.2. Simulation characteristics

When choosing the dynamic presentation to be used for spatial data, we took some psychological aspects into account.

(a) If spatial data are to be processed (as is the case in this displacement regulation), we know that analogic presentation of information is simpler and more reliable than digital presentation: it is easier to process spatial data in a visual mode than in an alphanumeric one. Moreover, in a continuous process such as ours, the importance of the temporal parameters makes this presentation more efficient.
(b) When concerned with the amount of information to be processed the tendency is to try to achieve "realism" and thus exhaustivity (for example in Computer Aided Design). We are also aware of opposite tendency, in which only "just necessary" information (from the analyst's point of view, of course) is presented. From our point of view, both of these extremes are naive and biologically

unscientific: the real problem is to present the human operator with isomorphic situations for his processing methods. The extraction of the most relevant cues (which is very difficult for the operator to do without any assistance) implies a certain amount of modelisation of these processing methods. This type of schematisation should be distinguished from the schematisation of mental images (Piaget and Inhelder, 1966) or from the schematisation of operative images (Ochanine, 1964) involved in cognitive regulation of action, and planning in particular. It is now well known, that mental images have the function of simplifying or even modifying reality, and as such can easy be used by the operators.

(c) The last aspect concerns the content of the graphic information. A continuous process is characterised by the continuous variations of the motion parameters. Therefore (and this remark is related to the first), cognitive processes are more easily used on states than on transformations. We are therefore encountered with the problem of needing discreet units for optimally continuous processes, to make the state sequencies compatible with the processing (principally from the temporal point of view) on one hand, but not modifying the processes on the other. This question is of much importance in this type of situation. Thus we did not construct driving simulations with slow continual motion, but we used position sequencies, with an image change about every 3 seconds.

3. ILLUSTRATION OF OUR METHOD - EVALUATION OF 2 MODES OF PRESENTATION OF GRAPHIC INFORMATION

3.1. Aim of the experiment

Dynamic aids, in maritime navigation, generally take two different modes of ship evolution into account: the alphanumerical (digital) mode, with parameters for speed, helm, heading, and so on; the analogical mode, such as the cartographic view of the radar screen. The data supplied by these aids are used in addition to the navigator's direct observations of his surroundings although the two may clash. We have therefore considered the importance of evaluating the respective quality of these different modes of data presentation in the same task, and of comparing graphic modes.

In this context, Bertsche et al. (1980) simulated ship steering in a curved channel, with three different modes of data presentation. They called them: digital, graphic (near the radar screen), and perspective. They obtained the following results: the quality of the trajectories derived by the graphic presentation was the best; perspective presentation led to a slight difference in performance although the ship did remain in the channel; whilst digital presentation, resulted in numerous "bad" trajectories (leaving the channel at the bend). These results show the graphic modes (graphic and perspective of the authors) to be superior. This fact is probably due to the similarity of this presentation with cognitive processes which are involved in spatial data processing. Therefore, neither of the authors took the subjects' processing methods into account; nor did they try to combine the different modes of presentation. Thus, we reconducted Bertsche et al. 's experiment, combining each graphic mode of presentation with digital data.

3.2. Description of the experiment

We simulated the steering of a ship in the channel of Antifer, which is Le Havre's oil terminal. This channel is 30 kilometres long, but the experiment was only concerned with the first bend area: each simulation lasted about 20 minutes. The starting point was always the same: at the North of the mid-channel line. This experiment allowed us to compare 2 modes of graphic data presentation: cartographic mode and perspective mode (see Figure 1); 3 groups of subjects: non professionals, pilots from Marseille and Le Havre; 2 conditions of stream: strong or weak. The subject was

always given digital data, and shown only one graphic mode of presentation. He carried out 2 trajectories under each condition of stream, but when starting he did not know their characteristics: he had to steer the ship in the best way possible.

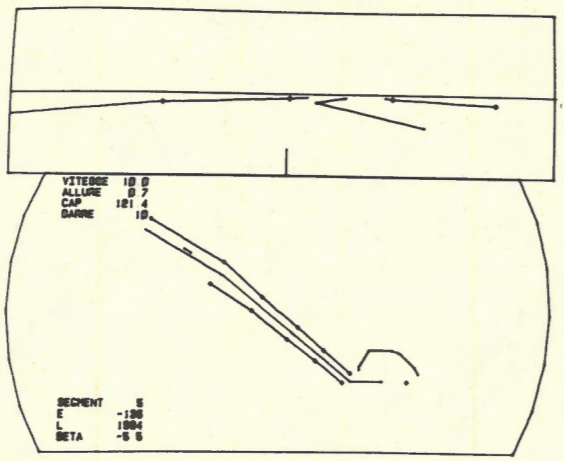


Figure 1 The computer screen with the data presentation modes. In the lower part (cartographic mode) the northern and southern limits of the access channel to Antifer port are shown by lines with the buoys; the mid-channel line is also drawn; the vessel is represented by a small mobile segment.
In the upper part the channel is represented by a perspective mode, wnere the vertical line corresponds to the bow of the ship, this is fixed in the centre of the screen.
The alphanumerical information concerns the bridge parameters (speed, rate, heading, helm) and the steering parameters of the Radio Electrical Aid System; these are: the segment number, the distance from mid channel, the distance to the following couple of buoys, drift angle.

This type of experiment allows us to compare two modalities of displacement perception and control. One of them does not exist in everyday life: in the cartographic mode, the subject sees himself moving in the space. This "decentralisation" phenomenon, however, does occur in localisation tasks for instance, where the subject has to designate his location on a map (Peruch, 1980). Since the graphic mode of presentation supplies fairly complete information about the surrounding space, the anticipation of the trajectory can be facilitate. On the other hand, the perspective mode of presentation gives a more realistic visual field, but more sensitive to variations; moreover, the similarity of the presentation with the scene from the bridge can render it more compatible with cognitive processes involved in trajectory control.

3.3. Principal results

The subjects' performance in controlling the ship's displacement was described using certain parameters which were supplied by the 2 software packages. The first, worked on the experimental computer (Hewlett-Packard 9825), and allowed us to plot the graph of the paths of a definite sample; the second, using a Hewlett-Packard 9845 computer, produced statistical and graphical results necessary for the analysis of some factor effects. We have chosen to discuss the results concerning the two "extreme" groups: non-professional subjects, and pilots from Le Havre-Antifer.

3.3.1. Performance analysis

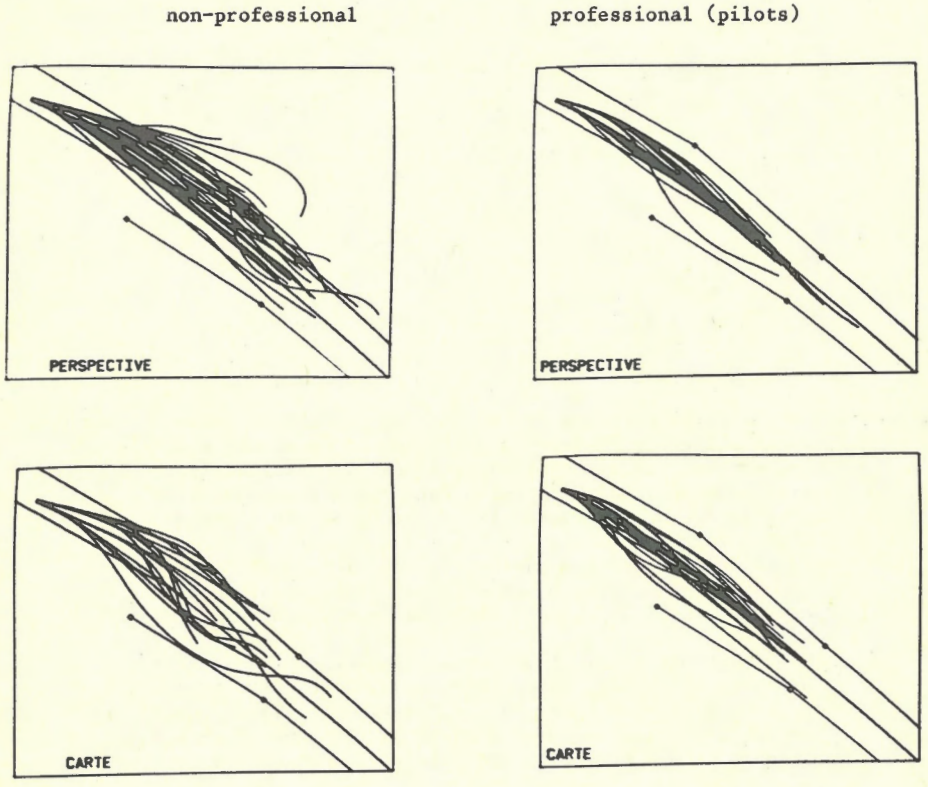


Figure 2 Examples of trajectories (non-professional subjects and pilots of Le Havre). The part of the channel in the experiment (first bend area) has been enlarged.

Trajectory paths (see Figure 2) were more regular and less dispersed for the pilots: their ability to extract relevant cues allowed them to anticipate manoeuvring and therefore to steer the ship correctly through the trajectory. This was not the case for the non-professional subjects who were more or less spectators of the consequences of their manoeuvring, and often could not avoid leaving the channel. The perspective one was however the most efficient of the different modes of presentation. We have shown that it is important to give presentations isomorphic to human

processing methods. Although within this mode of presentation few indications are given to the operator, it facilitates his anticipation of the trajectory, and it allows the use of speed gyration: speed is perceived by the perspective mode and at the same time its value is supplied by the digital mode. The superiority of the perspective mode of presentation found here, can be explained by the fact that the data available to the operator are more complete due to the association between the other modes of presentation. Most subjects were upset by the effect of the stream, though it did not affect the ship manoeuvring capacities (because its action is reduced to a translation movement); the pilots of Le Havre who knew the harbour were less disturbed than the other subjects and were capable of correctly anticipating the effects of the stream.

3.3.2. Manoeuvring relevance

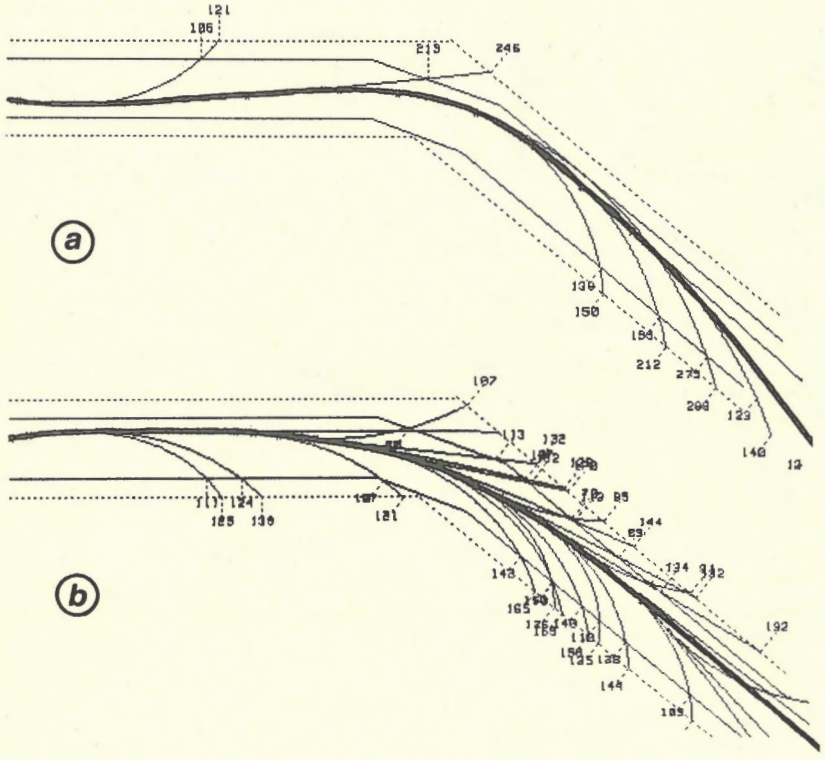


Figure 3 Channel "exit time" (in seconds) for two different subjects. The thick line corresponds to the trajectory actually carried out.

The subjects steered the ship by using helm and speed instructions, which were then simulated by the experimenter using his computer keyboard. A good estimation of manoeuvring relevance was, for example, the number of instructions necessary to keep the ship in the channel; figure 3 shows the variability of this number between the different trajectories the subjects produced: some of the subjects (see for example Figure 3b) used three times as much instructions as others (see for example Figure 3a). It was also possible to calculate how long the ship should have remained in

the channel if an instruction had not been given: the gain ratio of an instruction is given by the relation between the "exit time" of 2 successive instructions. This ratio depends mainly on the ship's position in the channel, and is relevant to the anticipatory quality of an instruction in relation to certain spatial constraints. The exit times are indicated on the paths corresponding to each instruction.

4. CONCLUSIONS

The experimental apparatus allowed us to evaluate the perspective mode of presentation, comparing it to the cartographic mode. In a simulated situation (where only apparatus data were presented), this mode of presentation appeared to be superior and seemed thus to be of considerable interest as a job-aid: data presented by perspective mode were isomorphic and thus compatible with data directly perceived from the bridge. By its simplicity (coding of relevant features only) and its similarity to the external space, this mode of presentation can really help the operator to "read" the surrounding space. The superiority of the perspective mode of presentation urges us to go on using it in our new experiments. By identifying the processing methods and their adequacy in solving the task, we will be able to anticipate the problems encountered by pilots. It will then be possible to prepare more complete training courses, and to include this new job aid into ships and data systems. These conclusions, of course, are not directly transposable to field situations; other studies using these results will be necessary.

As a last point, we want to emphasise the interest of using computer systems in large ship manoeuvring simulations. These systems are not so expensive and can easily be transported to the ships bridges. Moreover they can become good job-aids: they facilitate the operator's diagnostic, giving him information he is not always able to extract directly, and they also supply him with information about future states, helping him to anticipate decisions.

FOOTNOTE

[1]) This work was supported by a research agreement between the Laboratory and Société OPEFORM.

REFERENCES

Bertsche, W.R.,Cooper, R.B., Feldman, D.A., Schroeder, K.R. (1980). An evaluation of display formats for use with marine radio navigation piloting systems. United States Coast Guard. Washington D.C.

Bonnet, C. (1975). A tentative model for visual motion detection. Psychologia, 18, 35-50.

Brouard, J.,Deutsch, C., Routin, M., Cuny, X. (1979/1980). Etapes préalables à l'analyse de la conduite: l'exemple du pilotage portuaire. Bulletin de Psychologie, 33, 263-272.

Gibson, J.J. (1950). The perception of the visual world. Houghton Mifflin, Boston.

Gibson, J.J. (1979). The ecological approach to visual perception. Houghton Mifflin, Boston.

Gordon, D.A. (1965). Static and dynamic visual fields in human space perception. Journal of the Optical Society of America, 55, 1296-1303.

Lee, D.N. (1974). Visual information during locomotion. Perception: Essays in Honour of James J. Gibson. R. Mc Leod & H. Pick Jr. (Eds). Cornell University Press, Ithaca, New York. 250-267.

Nakayama, K.,Loomis, J.M. (1974). Optical velocity patterns, velocity sensitive neurons and space perception: a hypothesis. Perception 3, 63/80.

Ochanine, D.A. (1964). L'acte et l'image, problème d'ergonomie. XVIIe Congrès International de Psychologie Apliquée. Ljubljana. 81-88.

Pailhous, J.,Cavallo, V. (1982). Les effects spatiaux du mouvement: leur rôle et leur traitement. L'Année Psychologique, 82, 457-472.

Peruch, P. (1980). Localisation et orientation du sujet lors d'un dèplacement: étude de la performance. L'Année Psychologique, 80, 449-465.

Piaget, J., Inhelder, B. (1966). L'image mentale chez l'enfant. P.U.F., Paris.

PROCESSING TV INFORMATION AND EYE MOVEMENTS RESEARCH

Géry d'Ydewalle

Department of Psychology University of Leuven/Louvain
Belgium.

1. THEORETICAL INTRODUCTION

Broadbent (1958) postulated that human information processing is restricted by a limited capacity filter between the large variety of sensations we have and the attentive stages of input analysis. He claimed that both the visual and the auditory sensory systems function as parallel information-processing channels; that all environmental inputs (e.g., sounds and visual stimuli) can be received simultaneously. A precategorical analysis is performed: certain physical features are discernible (e.g., pitch and size), and others aren't (content, context, or meaning). The filter mechanism allows only one message at a time to pass from sensory memory into the attention system. For example, in an environment within which two human speech tracks run simultaneously only one can be attended to at a time. Broadbent claims that about 1.5 sec. is necessary to switch attention from one sensory input modality to another (Broadbent, 1971; Davis, Moray & Treisman, 1961; Moray, 1960), whilst others have estimated a much shorter span. Moray (1960) for example, indicates that 50 msec. is needed for very simple auditory stimuli (in our proposed studies, much more complex stimuli will be used). The degree to which our attention can switch between different inputs is also deemed limited. Treisman (1968) and others modified Broadbent's model considerably to incorporate findings indicating that some content material from the unattended channels do indeed break through to active attention. These models imply that although stimuli in unattended channels are severely attenuated they have not ceased to exist.

Neisser (1967) made a major central distinction between "preattentive processes" and "focal attention". In the preattentive stage holistic parallel processes use the stimulus information, arriving simultaneously, to construct the separate sensations involved. These result in rather crude impressions of the stimuli's properties (movement, general location, brightness, etc.) which have little or no effect on behaviour. This differs from the phase in which the stimulus information has become the focus of attention. Attention, according to Neisser (1967), is serial: Only one object can be attended to at any given moment, and each attentive act takes time.

When simultaneous processing of multiple sensory inputs is required, one often refers to "divided attention" situations. The serial-controlled mechanisms of attention produce divided attention limitations, but Schneider and Shiffrin (1977) and Shiffrin and Schneider (1977) have shown that these limitations can be bypassed when automatic-parallel processing is utilised. Automatic processing takes place in long-term memory, is triggered by specific inputs and operates largely independently of the subject's control. When automatic-attention processing is activated, it will not necessarily affect ongoing controlled processes. There is an obvious similarity between the distinction Neisser (1967) makes between preattentive and attentive processing and the distinction Shiffrin and Schneider make between automatic-parallel and serial-controlled processing. These are however also a number of dissimilarities, one of which has influenced our research proposal: contrary to Neisser (1967), parallel processing from multiple external stimuli to meaningful content and context is possible within the approach of Shiffrin and Schneider insofar as the automatic-

parallel processes for the appropriate multiple inputs are well learned.

One of the assumptions found in recent theories is that short-term memory has automatic and controlled processing and storage functions that, in some cases and especially with serial-controlled processing, compete for a limited capacity within the short-term memory. This conception can be contrasted with the more traditional theories that view short-term memory, now more commonly labeled "working memory" (Baddeley & Hitch, 1974), as having storage functions only. The more recent theories that agree upon the existence of demanding processes (processing and storage) that consume the available capacity and activities with no capacity trade-off (with parallel processing), have led Navon and Gopher (Gopher et al., 1982; Navon & Gopher, 1979, 1980) to propose, and to provide evidence for a multiple-resource allocation theory. In this approach, the human-information processing system incorporates a number of mechanisms, each having its own capacity. These capacities can be allocated among several processes at any given moment.

2. INFORMATION PROCESSING OF AUDIOVISUAL MATERIALS

The previous discussion presents issues which are important for research in audiovisual broadcasting and presentations on Visual Display Units (commonly called VDU investigations in human factors research). The implications of the earlier analyses Broadbent and Neisser made for audiovisual presentations are clear. At any given time, either the audio or the visual input is fully analysed. The number of input modes however increases, if the video display also contains printed material (e.g., subtitles) superimposed on a moving visual image or if presentation contains both speech and music. Switching attention between the separate input modes takes time and should therefore be avoided. If, however, the assumption is made that parallel processing of certain materials or multiple-resource allocations occurs increasing the flexibility within the human system then fewer inter-input interferences are likely to occur. Nevertheless, as the information load increases, the attentive system will eventually become overtaxed. The system will then only accept input from one source at a time, working as a single-input system.

Although film, television, and VDU research use perceptually rich material that is ecologically beyond the laboratory limitations of simple, single stimulus presentations, most of the research that has been undertaken has not been concerned with attention and processing issues. Our research focuses on the processes and factors involved in reading subtitles in films and the way in which the subtitles are related to the other inputs from the screen.

Television companies often import programmes and films in which a "foreign" language is spoken and which therefore need translating. Subtitling is used in several Western European countries including Belgium, the Netherlands, and the Scandinavian countries because it's cheaper than dubbing which is used in the Federal Republic of Germany, France, Greece, Italy, Spain and Portugal. Subtitling adds one input channel to audiovisual presentations. A large number of problems worth investigating arises when one seriously looks at what the subject is doing when watching television. The time used by the subject to read a subtitle can be influenced by his knowledge of the foreign language spoken. Children get into the habit of reading subtitles, and it is possible that this reading behaviour is automatically triggered off even when, as adults, they have mastered the spoken foreign language sufficiently. Understanding the foreign language or reading the subtitles can be considered superfluous for those parts of a movie which provide redundant information. The processing of information is needed to follow and understand the movie, switching attention from the visual image to either the audio channel or to the subtitle. This switching takes time and exploits loading capacities of the short-term or working memory. If a certain amount of parallel processing is possible, two input channels (e.g., the audio and visual tracks) could be entered into our cognitive system without too much loss of time and available memory space. A considerable amount of

research on eye movements in reading text materials has recently shown that reading a subtitle requires sequential focusing (Just & Carpenter, 1980; Rayner, 1978). Reading behaviour typically involves a sequence of eye fixations on words (above 150 msec. duration), which are preceded and followed by saccadic movements (about 30 to 50 msec. duration). A number of variables affect the duration and the sequence of the fixations (Rayner, 1983). If parallel processing between the several input channels is possible, the subject could process the subtitles on a superficial level by means of peripheral vision sufficiently to understand the movie. Our first research question therefore considers aspects of reading behaviour: Do sequential eye fixations and saccades occur when subtitles are read.

About 40% of the television output of the Dutch channel of Belgium (BRT) is from abroad, and almost all of it is subtitled (Muylaert et al., 1983). A rule of thumb is used for determining how long a subtitle is to be displayed. A subtitle consisting of 2 x 32 characters and spaces is displayed for six seconds. Shorter subtitles are scheduled proportionally. The origin of this rule of thumb is unknown. Psychologically speaking, it is surprising that presentation time is defined by the number of characters and spaces in the subtitle, if one assumes that normal reading behaviour occurs. The available literature on eye fixation suggests that one indeed reads words in such a situation and that fixation time and sequence are determined by the content of the words in the subtitle, and not by an arbitrary string of characters and spaces. On the other hand there is also the possibility that some form of superficial processing of the subtitles is done in peripheral vision (e.g., keeping track of the orthographic properties of the words). There is literature available in which the suggestion has been made that the orthographic lay-out of the individual characters within a word may be used to access the meaning of the words and the content of the subtitle (Massaro et al., 1980).

The BRT, in collaboration with Open University in England, have carried out the first pilot study on subtitles (Muylaert et al., 1983). Voge (1977) reviewed the subtitling versus dubbing debate and concluded that little empirical testing is available. Muylaert et al. (1983), constructed subtitles for "Dallas" extracts to test the hypothesis that the amount of attention paid to a subtitle depends on a number of variables: the use of one line versus two; the occurrence of unusual breaks between two lines or between two successive subtitles; and finally, deviations from the six-second rule (shorter and longer presentations). Variations in eye-movement patterns occurred with changes in type of subtitles and with the education level of the subjects. Generally speaking, viewers seemed to find it very difficult to avoid looking at subtitles. However, detailed analysis of the data was not possible due to technical limitations.

The quality of equipment in our laboratory has improved since the first investigations. While implementing the different components of our eye movement monitoring and registering equipment (DEBIC 80 + PDP 11/40), we extended the first study of the BRT (for a full account of our study, see Muylle, 1984). A German movie was used because of the subject's unfamiliarity with this language. The analyses of our data suggested a few conclusions that still need more detailed analyses. The eyes are on the subtitles at least for one third of the subtitled time (our subjects were first-year university students). During this time, our subjects did not show the typical eye movement pattern of reading behaviour (except with a few subtitles with one or two subjects). Subjects more commonly look at the moving image first and quickly jump to one or two keywords from the subtitle. The jump (or saccadic movement) is quite accurate, which may be explained either by parallel processing in peripheral vision of parts of the subtitle or by corrections during the jump. The last possibility is not likely as several studies suggest that visual acuity during a saccade is either absent or at least severely diminished (for more recent studies, see Haber & Hershenson, 1973, and Leisman, 1978). One key issue here concerns what is picked up in parafoveal vision: this should be investigated more carefully. Another finding of our study is that subjects spend more time on the subtitle when longer presentation times are provided (four-, six-, and eight-second rules were used in this study). Again, this may mean that the subjects do need more time to process the subtitle. However, it is also possible that subjects first look at the

subtitle, then go back to the moving image, and, if more time is made available (with the eight-second rule), jump back again to the subtitle. This cyclic pattern could explain the longer viewing time on the subtitle and also the almost complete absence of reading behaviour of our subjects.

3. SUMMARY

The purpose of our contribution was to make an inventory of the trade-offs between looking at the moving image and processing the subtitle. Does reading occur? Is there successive scanning from the pictures to the subtitles and from the subtitles back to the pictures? Could both forms of information be processed simultaneously with no loss of understanding of the flow of information in the moving image? What information is captured in peripheral and parafoveal vision, and how is it used in the purposive sequence of eye movements?

REFERENCES

Baddeley, A.D., Hitch, G. (1974). Working memory. The psychology of learning and motivation, Vol. VIII. Bower, G.H. (Ed.). Academic Press, New York.

Broadbent, D.E. (1958). Perception and communication. Pergamon Press, Oxford.

Broadbent, D.E. (1971). Decision and stress. Academic Press, London.

Davis, R., Moray, N., Treisman, A. (1961). Imitative responses and the rate of gain of information. Quarterly Journal of Experimental Psychology, 13, 79-91.

Gopher, D., Brickner, M., Navon, D. (1982). Different difficulty manipulations interact differently with task emphasis: Evidence for multiple resources. Journal of Experimental Psychology: Human Perception and Performance, 8, 146-157.

Haber R.N., Hershenson, N. (1973). The psychology of visual perception. Holt, New York.

Just, M.A., Carpenter, P.A. (1980). A theory of reading: From eye fixations to comprehension. Psychological Review, 87, 329-354.

Leisman, G. (1973). Oculo-motor system control of position anticipation and expectation: Implications for the reading process. Eye movements and the higher psychological functions. Senders, J.W., Fisher, D.F., Monty, R.A. (Eds.). Hillsdale, New Jersey.

Massaro, D.W., Taylor, G.A., Veneszky R.L., Jastrembski J.E., Lucas, P.A. (1980). Letter and word perception. North-Holland, Amsterdam.

Moray, N. (1960). Broadbent's filter theory: Postulate H and the problem of switching time. Quarterly Journal of Experimental Psychology, 12, 214-221.

Muylaert, W., Nootens, J., Poesmans, D., Pugh, A.K. (1983). Design and utilisation of subtitles on foreign language television programmes. Theorie, Methoden und Modelle der Kontaktlinguistik. Nelde, P.H. (Ed.). Dummler, Bonn.

Muylle, P. (1984). Oogbewegingsregistratie bij het bekijken van anderstalige ondertitelde televisieprogramma's. Niet-gepubliceerde licentiaatsverhandeling. Leuven.

Navon, D. Gopher, D. (1979). On the economy of the human information processing system. Psychological Review, 86, 214-253.

Navon D., Gopher, D. (1980). Task difficulty, resources, and dual task performance. Attention and performance, Vol.VIII. Nickerson R.S. (Ed.). Erlbaum, Hillsdale.

Neisser, U. (1967). Cognitive psychology. Prentice-Hall, Englewood Cliffs.

Rayner, K. (1978). Eye movements in reading and information processing. Psychological Bulletin, 85, 616-660.

Rayner,K. (Ed.) (1983). Eye movements in reading: Perceptual and language processes. Academic Press, New York.

Schneider, W., Shiffrin, R.M. (1977). Controlled and automatic human information processing: I. Detection, search, and attention. Psychological Review, 84, 1-66.

Shiffrin, R.M., Schneider, W. (1977). Controlled and automatic human information processing: II. Perceptual learning, automatic attending, and a general theory. Psychological Review, 84, 127-190.

Treisman, A. (1968). Strategies and models of selective attention. Psychological Review, 76, 282-299.

Voge, H.(1977). The translation of films: Subtitling versus dubbing. Babel, 23, 120-125.

FROM SURFACE FORM TO THE STRUCTURE OF THE INTERFACE - STUDIES IN HUMAN COMPUTER INTERACTION AT INRIA

Pierre Falzon

Institut National de Recherche en Informatique et en Automatique, Le Chesnay
France

The central problem in man-machine interaction is the compatibility between two elements:

(a) the operator, and his physical, perceptual and cognitive characteristics;
(b) the machine, and its different aspects: dimensions and lay-out, information coding and information structure.

At a first level, the designer must endeavour to attain a certain compatibility between the physical characteristics of the system and the physiological and perceptual characteristics of the human. At this level, the designer is concerned with work place dimensions and general lay-out, information visibility, etc. This field - ergonomics - is now very well established. At a second level, the designer's task concerns the compatibility of the system with the elementary operations performed by the operator. These operations consist of acquired schemes, which can be sensori-motor, procedural and /or anticipatory. This rule-based behaviour must be matched by appropriate surface aspects of the machine. The designer has to choose, for each sub-task, the optimal way to encode information in order to facilitate the use of these schemes. To give an example, for a tracking task, different types of displays will be studied (pursuit, compensatory, predictive, analogical, pictorial, etc.). The studies of stereotypes (which are sensori-motor or cognitive routines) belong to this second level of compatibility. This is the field of human factors. Finally, at a third level, the designer has to take two fundamental human activities into account: information processing and mental representation. The relevant questions become: which information is processed? Which variables are elaborated by the operator? Which heuristics, which reasoning algorithms are used to reach the goal? What are the characteristics of the mental representation of the system? At this level, studies of the knowledge activities are fundamental. On the machine side, the designer is no longer concerned with information encoding, but with information structuring. This is what cognitive engineering is about. The global compatibility of a man-machine system can be achieved only if compatibility exists at each of these three levels.

As a matter of fact, the available body of knowledge is very unbalanced. We know a lot about ergonomics, quite a bit about human factors, and not much about cognitive engineering. The aim of the Ergonomic Psychology Project at INRIA is to contribute to the development of knowledge on the cognitive activities of the human elements of the systems: this knowledge can be used to design machines adapted a priori to their users' functioning logic. Our work is then clearly focused on the third level of compatibility described above. However, it is sometimes difficult to have a clear-cut separation between the different levels, especially between the second and third one. In fact, surface form and deep structure interact in several ways. The aim of this text is to present some aspects of these interactions, which will be illustrated by examples from different studies we have conducted. We are concerned with all problems related to human-computer interaction, whatever the domain of application: process control, office work, data base interrogation, programming, etc.

1. THE DESIGN OF INFORMATION DISPLAYS

I will begin with an example from a field which we have studied for quite a number of years, Air Traffic Control (ATC), and which clearly differentiates between the second and third level of compatibility. The main goal of the ATCers' task is the security of the aircraft they control. In order to achieve this goal, ATCers have to process information given by different sources: the flight plans and the radar screen are the two main ones. The radar screen (see fig. 1) displays information concerning:

(a) individual airplanes
(b) the present state of the situation

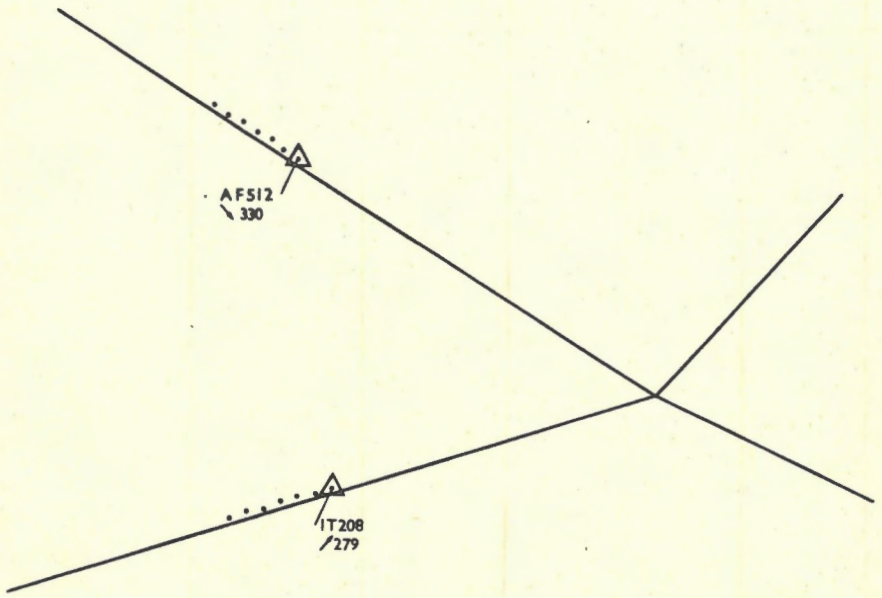


Figure 1 A simplified radar image

This display is structured in terms of objects and properties: the objects (the airplanes) take on some properties (position, heading, speed, flight level, attitude, etc.). Each object is independent as far as the image of the overall situation is concerned. On another hand, the analysis of the operators' information processing activity has shown that the reasoning process was not centered on the present state of each airplane, but on the future separations (vertical and horizontal) between airplanes considered pair by pair (Bisseret and Girard, 1973). The mental representation of the the traffic situation is structured in terms of variables and relations between these variables (Lafon-Milon, 1981). The relevant parameters are not objects and properties, but two variables (the vertical and horizontal separations) and their relation: each of these variables is the result of the processing of more elementary data, i.e. of object properties. For example, in order to evaluate the future horizontal separation, the operator has to process information relative to the present positions and speeds of two airplanes. The operators' task is then, starting from the information displayed by the machine, the elaboration of a mental

representation, the structure of which differs from the machine representation. In order to reduce the gap between the two representations, we have proposed (Falzon, 1982a) a new display, structured in terms of variables and relation. Figure 2 presents the same situation as in figure 1, but with a different display structure. Along the abscissa, the values of the horizontal separation (in nautical miles) are plotted. The ordinate shows the values of the vertical separation (in thousands of feet).

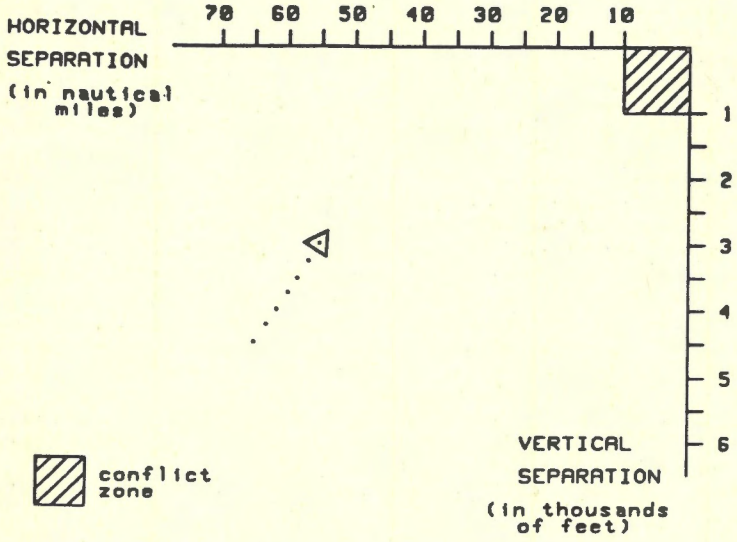


Figure 2 Representation structured in terms of relation

The triangle indicates the simultaneous separation values for the pair of aircraft of Figure 1. The dots behind the triangle represent the evolution of the separation. They allow the display of the direction and speed of variation of the separation. The controller's task is then to estimate whether the triangle will get inside the conflict zone at the intersection of the axes. The issue to be stressed here is that the surface form proposed in Figure 2 is probably not the best way, and certainly not the only way to formalise this display structure. A multitude of other graphic modalities can be (and actually have been) thought of. This is in conformity with what was said earlier, i.e. that having chosen a display structure compatible with the operators' mental processes (third level of compatibility), the designer still has to choose an appropriate surface form for this structure (second level of compatibility). In the above case, one may presume for instance that other surface modalities might facilitate the extrapolation task.

The lack of compatibility between man and machine representations can have very negative consequences, as a study of the problem solving activity of the Paris subway traffic controllers has shown (Senach, 1983). In this system, the controlled objects are bi-dimensional: one dimension has to do with the planning of the train movement, the other one refers to the drivers assignment. These two dimensions are independent: a driver does not always drive the same train all day long, and a given train may be used for different purposes within a day. Thus, incidents differ according to their consequences: when a train stops, it can cause delay for the train, for the driver or for both. Several studies have been conducted (simulations of problem solving with experts) and their results have pointed out that:

(a) although the experimentations did not introduce any temporal constraints in their design, the analysis of the problem is not very precise: some of the important variables (relative to the drivers) are not processed.
(b) the operators process two incidents belonging to two different types of problem in the same way: in some cases they only look for solutions which can reduce the delay of the train, not taking other variables into account, and all the solutions chosen generate a delay for the drivers.

These results have been interpreted in terms of problem space reduction. In other words, the operators process a problem simpler than (and different from) the real one, by failing to analyse all of the relevant parameters. What is especially impressive is that, although the operators have had a long experience with the system, they are still not experts. This simplification of the problem can be explained, in particular, by the fact that, in the present system, one of the two important variables is more salient than the other. The trains positions are displayed in real time on the control panel and the most important document is the graphical representation of the process showing all the theoretic states and train positions. Data about drivers are only supplied on an alphanumeric list providing all the assignments for each driver. This representation system makes it difficult to combine the information relative to the two dimensions of the controlled objects, especially because of the time constraints which hold for the decision making process. This experiment clearly shows the direct effect of a display structure on the operators' processes. The problems that are met in process control are often caused by a lack of consistency between the displayed information and the cognitive processes. The operator is forced to switch from the machine representation to his own (Bisseret and al., 1979). The situation for the operators of the Paris subway is worse: the structure of the display has made it impossible for the operators to build an adequate representation of the problem space.

2. INTERACTIVE LANGUAGE STUDIES

Language is a question of fundamental importance in the research we undertake, and this for several reasons. First, because language is, by necessity, the only way to interact with a computer, whatever type of language (programming language, command language, natural language), symbols (alphabetical, pictorial) the medium of communication (keyboard, mouse, joystick, etc.) are used. Second because language is the external representation of a cognitive activity. As such, the analysis of language use can provide us with important clues as to the mental processes involved in a given task.

2.1. Command languages

One branch of research has tackled the question of command languages. There are numerous studies of existing command languages, we have evaluated some of them. The results are very often quite negative but instructive. To give an example, Escarabajal (1982) has evaluated the command language of a graphic editor. The system uses a graphic language: the possible actions and objects are represented by icons. In one of the experimental conditions, subjects had to match the icons with their definitions. Even in this situation (which represents the "easiest" condition) a considerable number of incorrect associations were made. The results show that subjects experience much difficulty in understanding symbols that represent actions and have far less trouble associating icons and objects with each other. The author concludes that the command language should use icons to refer to objects, and words to evoke the actions performed on these objects. This first example shows that it is impossible to define the vocabulary of a command language without taking the fundamental cognitive processes of the language user into account: in the previous example for instance, the recommendations of the author (icons for objects, words for actions) are probably closely related to the way these concepts are stored in memory. In

other words, the solution to the problem is not to be found in re-drawing the icons (which would be a typical human factors approach), but rather in choosing a different representation system, more adapted to the cognitive structures of the users. Several studies have explored the problem of designing command languages. Research on naming has grown rapidly in the recent past, motivated by changes in the users population: prospective users are no longer computer specialists, willing to adapt to the machine. Most designers are aware of this situation, and, in the hope of facilitating the learning and use of the computer functions, the names of the commands have often been selected among the general vocabulary of the (English-speaking!) user. This inclination is motivated by the assumptions that commands will be more or less self-explanatory for the users (they will evoke concepts that these already possess), and that they will be remembered (or even guessed!) more easily: by choosing an everyday language, the designer hopes to build a friendly interface. That may indeed seem a reasonable perspective. Alas, things are not that simple.

The problem of naming is, at a first level, to be considered a classic human factors problem, dealing with information coding. At that level, the relevant questions concern the optimal choice among different modes of representation (for instance: icons or words?), different sets of symbols, etc. At a second level, the problem of naming is a problem of cognitive engineering, i.e. a problem of compatibility between the mental processes of the user and the functioning logic of the machine.

What is the situation for the naive users? They have some knowledge of the operation they can perform when the task is done manually, i.e. without the computer. Associated to these operations, they know a set of words, which they use to refer to these operations. The problems are then:

(a) are the meanings of the words used in a given context (the manual task) easily extendible to a different context (the computer domain)?
(b) are the operations in the manual task similar to the operations of the computer version of the task? This question can be extended to include:
    - is it possible to establish a one to one mapping between the manual operations and the computer functions?
    - are there any computer functions that have no equivalent in the manual task?

What is now the situation for the experienced users? In addition to the knowledge of the manual task, they have some knowledge of what computers are in general, and of the functioning of one or several computers.

D.L. Scapin, in a series of experiments, has studied the use of computer commands in restricted natural language by naive and experienced users. In one experiment (Scapin, 1981), subjects (with or without experience in the computer domain) had to learn to use a text editor, and were submitted to a cued recall experiment one week later. During learning, subjects were given command definitions, varying two factors: redundancy (the command word was, or was not, repeated in the definition) and context (operational, i.e. related to the task, or functional, i.e. related to the computer function). Some of the results of this study will be presented here.

First, redundancy is more helpful to subjects with experience in the computer domain than for naive users. This can be explained by the competition (for the experienced subjects) between the new command words and the command words learned on other systems. Redundancy is useful for the memorisation of these new associations.

Second, experienced subjects are better in recalling the possible functions of the system, although they are not always able to recall the commands associated to the functions, and they are able to rephrase the command definitions. Naive subjects do not memorise as many computer functions, but the recalled functions are memorised with their labels. Recall is more difficult if the formulation of the cue differs from the definition given in the learning phase.

Third, and surprisingly, naive subjects' recall of commands is more extensive if the

commands belong to a technical vocabulary. This phenomenon can be explained in the following way:

(a) for the experienced subjects, words only represent the surface form of an underlying concept. Through their interactions with computers they have built an ability to conventionalise. They are able to differentiate the concept, which is stable, from the command, which varies according to the systems. Their only problem is to identify the concept, and then to try to avoid errors related to the fact that, in other systems, this concept is not evoked by the same command terms, i.e. errors related to the competition between different surface forms.
(b) for the naive subjects, when command terms belong to the general vocabulary, the concepts that are evoked belong to their general knowledge, and interfere with their representation of the computer function. This problem does not exist if commands are formulated in a technical language. The phenomenon is also influenced by the fact that the computer concepts are not well established: naive subjects are still very dependent on the surface form of the function.

The author suggests an interesting method: in order to facilitate the learning of a command language by naive users, the operational definitions of the commands should be given first, and then the functional definitions. In that way, the learning of the computer domain will be based on the knowledge of what the subject already knows. As Scapin notices, this is quite different from the way user's manuals are generaly structured.

Another experiment (Scapin, 1982) has focused on the effects of having the subjects themselves name the commands on learning. Subjects have been tested in two conditions: command creation vs imposed commands. Two other variables were used: structure (in the creation condition subjects were, or were not, given a structuring rule; in the imposed condition, commands were, or were not, structured), and number of commands (31 or 13).

This experiment has provided a lot of interesting results concerning the usefulness of structure in the acquisition of the command language, the effects of the number of commands, and the spontaneous syntactical patterns generated by the subjects. Here we will focus on another result, concerning the effect of the generation of the commands by the subjects. The recall performance is better with computer commands created by the subjects than with an imposed language. This generation effect is stronger when a structuring rule is supplied.

Concerning the language designer, these studies have interesting conclusions. Language designers are in the worst position to give an opinion on the language they have built. First, because they are all computer specialists, and, as such, have acquired a knowledge of the possible computer functions. Second because, through their interactions with different computers, they have developed an ability to conventionalise, i.e. they are able to learn new and arbitrary relationships between a computer function and a surface form quickly. And third, because they benefit from the generation effect: the names given to the commands are their own.

2.2. Operative languages

The research we have presented assumes that in the foreseeable future, most man-machine interfaces will still use a restricted dialect. The question is then to study appropriate restrictions, which would not hamper the communication. This is the approach followed in studies of computer commands in restricted natural language. However, a second approach is possible. Instead of studying specific restrictions of natural language, why not study the natural restrictions of specific languages? When the operators of a system have to communicate verbally, they tend to build operative languages, molded by the characteristics of the task and its objective. These languages are restricted, as compared to natural language, in a number of domains: vocabulary, syntax, field of discourse are a few. They have the powerful capabilities of natural language (they can give commands, request or give

information, comment on the situation, etc.), while avoiding its possible ambiguities. In fact, even subjects allowed to use natural language in typing instructions for a computer have been observed to restrict the way they express themselves, i.e. they switch to an operative dialect (Gould et al., 1976). In that perspective, natural language cannot be considered a natural command language.

The study of operative languages would be helpful for the design of computer command languages, in two different ways:

(a) for a given domain, by providing the vocabulary and the forms of expression appropriate for the computer version of the task;
(b) whatever the domain, by providing indications for the elaboration of these languages, and particularly on the natural rules of restriction.

This approach will be illustrated by the study of a specific language, used by Air-Traffic Controllers (ATCers) in their communications with aircraft pilots. A first study (Falzon, 1982b, 1983a) has shown two important points:

(a) this language uses a limited number of words. Moreover, it is possible, starting from this lexicon, to design an even more restricted vocabulary, still allowing a large number of messages to be taken into account.
(b) most messages begin with a "command" word: this command word is sufficient for the categorisation of the messages. Moreover, in each category, the possible constituents of the messages are highly predictable.

In other words, as soon as the command word is understood by the expert, a specific body of knowledge can be activated, allowing expectations on the following information. Schema theory provides a framework for this process. The messages of a category can be considered as a list of different instances of a common underlying schema, as different actualisations of a single schema. Each schema is a pre-defined frame, with slots requiring to be filled with specific pieces of information. The understanding of a message can be described as a process which is first data-driven (a schema is activated by some schema-associated word) then conceptually-driven (the schema is then instantiated, i.e. the slots of the schema are filled with the information from the message). This analysis has been developed in a second study (Falzon, 1983b, 1984). The messages emitted by the controllers (in a second corpus of ATC communication) have first been categorised, and a method of schema abstraction has been devised. The method is based on the fact that, in each category, the messages vary in different ways. The surface form (vocabulary, form of expression) varies, of course, but also the way the schema is instantiated, i.e. the content: two messages can be paraphrases, or can have a varying amount of overlap in their meanings. The analysis of the variation in surface form and in meaning allows:

(a) a description of the schema of each category of message,
(b) the elaboration of a dictionary of words, which has the following characteristics:
    - it does not include all the words used, but only the necessary words (laconism);
    - the word definitions are phrased in terms of the schematic knowledge previously defined, i.e. are domain-oriented (functional distortion).

Laconism and functional distortions are two general characteristics of operative representations, in the sense of the theory of operativity developed by Ochanine (1981), as opposed to cognitive representations. A cognitive representation tries to give the most accurate, objective description about an object; an operative representation, on the contrary, is biased and incomplete, tailored to fulfil a given task (different examples of the operative attitude of subjects involved in different types of task are given by Bisseret, 1983).

Among the words of the dictionary are schema-associated words, i.e. words that evoke a specific schema.

The schematic knowledge has been implemented in programs, which were tested using another corpus of ATC communications. The first result concerns the success of the approach. Although the system has virtually no syntactic abilities (the only syntactic clue being that messages generally begin with a schema-associated word), and although the vocabulary is indeed very restricted (the programs only know of 75 different words, 33 of which are schema-associated), it can process 78% of the controllers' commands. This is especially impressive when one considers that a number of failures in understanding are provoked by limitations of the knowledge base: some words, categories of messages or message modalities did not appear in the corpus that was used to define the dictionaries of words and schemata. A larger corpus would certainly increase the system's performance.

A second result concerns the analysis of the difficulties met by the system in understanding. The most frequent single type of errors (32% of the understanding problems) were caused by omission of the command word. In those cases,since the system expects to find a schema-associated word at the beginning of each message, no schema is evoked and the system fails to understand. Ellipses of the command word occur wnen the controller knows that the dialogue or situation context is so constraining that the pilot is expecting a given category of message. In other words, ellipses occur when the context itself evokes a schema. A conclusion based on this observation is that understanding would be improved by two different devices:

(a) a communication analyser, which would keep track of the recently evoked schemata, and which would diagnose dialogue-driven ellipses.
(b) a situation analyser, which would be based on a description of the flight plan as a hierarchy of scripts. Knowing the script being performed, the system can infer the schemata that can or cannot be evoked. These expectations can guide the comprehension of the messages, and allow the diagnosis of script-driven ellipses.

2.3. Understanding and debugging programs

Programming languages are another way to give commands to a computer. Criteria such as duration of program design and development, ease of maintenance (or debugging) and ease of learning have become very important for the evaluation of these languages. It is now obvious that facilitating the comprehension of computer programs decreases the duration and cost of programming, and reduces the programmer's workload. A recent experiment (Detienne, 1983) has investigated the activity of understanding computer programs. The experiment (conducted on expert programmers) included a debugging task, followed by a recall task, in which the subjects were asked to rewrite the program as accurately as possible. The hypothesis was that understanding would involve two different kinds of processes: data-driven processes, guided by the program code, and knowledge-driven processes, guided by the expertise of the subjects. An interesting example of the interaction between both processes will now be described. During the debugging task, subjects were asked to verbalise what they were reading or thinking as much as possible. One of the errors they had to diagnose was of the following form:

```
GAP = A + B
```

The correct instruction should have been:

```
GAP = A - B
```

This error was sometimes very hard to spot. In fact, what happens is that when subjects read the word "gap", a body of knowledge about what gaps generally are is evoked, and provokes the inference that a subtraction will follow. This expectation leads to inaccurate perceptions of the + sign. For one of the subjects, the inference was so strong that he read aloud "minus" instead of "plus" three times in a row. In the recall phase of the experiment, two different kinds of errors appeared. Both deserve to be mentioned. The first type of errors concerned the name given to

the increment variable in a loop instruction. In the initial program, the increment was systematically named "j". Subjects frequently recalled "i" instead of "j". This confusion is particularly interesting when compared to the second type of recall error. Distortions appeared on the names of some variables. These variables had been given arbitrary names by the original writer of the program. During recall, a number of subjects changed these into meaningful terms, i.e. words related to what the variable stood for.

In other words, distortions in recall do appear, but for different reasons, and certainly not randomly. Increment variables are very often named "i" or "j" (by convention: there is of course no obligation to do so). The name of these variables is totally arbitrary and plays no role whatsoever in the representation of the program logic. Thus, changing "j" for "i" is probably an indication that the surface form has not been memorised at all, and that only its category (i.e. "increment") can be accessed, the subjects then choose among the usual codes. The reverse is true for the labels of the other variables. Their names are helpful in building a representation of the program, and if they do not seem clear enough, the subjects spontaneously code them appropriately. The construction of a mental representation of the way the program operates is then facilitated if meaningful names are given to the meaningful variables.

3. STUDIES IN SYSTEM DESIGN

It is one thing to choose the vocabulary of the commands of a system, it is another to define the necessary functions. In other words, it is worth studying the names of the commands, but it is certainly necessary to study the operations these commands will initiate. Very little research addresses this topic. In most cases, designers ask for some help, once the system has already been built, and only to adapt the surface aspects of the system. Designers assume that the functions of the system correspond to operations that are meaningful to the operators. One can very much doubt the validity of that point of view, especially in the prospect of designing intelligent systems, capable of efficiently assisting the users. The functioning logic of the machine, based on the designer's logic, differs widely from the logic of use (Richard, 1983). The design of adapted systems requires the availability of a model capable of describing the processes of the operators, namely: what is the users' representation of their activity, and how do they plan their actions? The goal of the research some of us are conducting is therefore focused on the structure of the planning activity of experienced office workers. Two hypotheses are made. The first hypothesis is methodological: when operators speak about their activity, they often begin their explanations by expressions like: "I send the invoices", I take care of the contracts". These expressions generally refer to a macro-action composed of more elementary subtasks, that they can express in more detail if they are requested to. Thus, the verbalisations are modelled by the way operators program their actions, by the existence of a hierarchy of goals and sub-goals. A second hypothesis is that subjects involved in a given class of activities do not possess a set of procedures corresponding to the whole set of possible tasks: they adapt, they combine a restricted number of procedures to meet the constraints of the situation they face. For this reason, the analysis of the operators' planning activity may allow the description of a set of elementary actions that have the following characteristics:

(a) they are seen as sub-goals by the subjects;
(b) they are common to a number of activities;
(c) they are composed of a body of more elementary actions relatively consistent and independent of the task context.

This study, very much inspired by different works in AI, implies the definition of a formal model which will be used to describe the actions at any level. One goal of the analysis is then the elaboration of a frame of representation and of its

necessary slots. Some of these slots can already be predicted: for each action, the frame must allow the specification of pre and post requisites, of the subgoals of the action, of their status (compulsory, optional, compulsory under a condition), of their ordering (procedure or free order), etc.

A first research project in that area has been completed (Sebillotte, 1983a, 1983b). Four different types of office work have been studied. The results are very encouraging. From a methodological standpoint, it seems relatively easy to make the structure of the planning activity apparent. For example, one of the subjects stated: "My main job concerns the medical accounts". To the experimenter question "What does that mean?", she answered "They must be typed and mailed". Typing and mailing are for the subject the two sub-goals of that specific activity. These sub-goals are independent in the sense that one of them could be done by another operator, but typing is a pre-requisite for mailing. A limited set of actions seems to be sufficient to describe the operators' activity, whatever the specific activity of the division. The same goals and sub-goals are phrased by the subjects. The differences lie in the objects on which the actions are performed (contract, invoices, forms, etc.), and on some characteristics of the planning activity:

(a) In some cases the objects influence the choice and number of sub-goals. One of the goals was for example "order something". The sub-goals are:
- if the object ordered is "stationery": fill in a form, keep a copy.
- if the object is a large piece of expensive equipment: fill in a form, ask for a signature, keep a copy.
- if the object is a book: check availability, fill in a form, and keep a copy.

(b) Some actions are characterised by a fixed order of subgoals, others are more flexible. In those cases, although the same sub-goals appear for all subjects, and although each subject may use a constant procedure, procedures may vary across subjects.

Another interest of this study lies in the analysis of the way the activity is verbalised.

The first point to notice is that the subjects use a variety of forms of expression to refer to the same underlying action concepts. This does not mean that a command vocabulary is impossible to define, but that defining the actions and defining the commands are two different matters.

The second point is the vocabulary used by each subject. One interesting observation in that respect concerns the variation of the level of meaning of some words. Here is an example of this phenomenon: the word "send" is used to refer to the MAIL action, if the document has already been typed, verified, signed, etc. But the same word "send" can be used to refer to a whole procedure, SEND, i.e. the set of sub-goals TYPE + VERIFY +SIGN + MAIL. Different interpretations of that behaviour can be thought of; we will propose one. The word "send" chiefly refers to the MAIL sub-goal; it can indifferently be used to refer to the SEND macro-action or to one of its subgoals because MAIL represents the fundamental feature of the procedure. In fact, any sequence of actions that includes the mail sub-goal can be considered an instance of a SEND macro-action. To a certain extent, a procedure consisting of a single MAIL sub-goal could be considered a minimal instance of the SEND action.

In that perspective, one design problem is the fact that the external representation of the actions, i.e. the language used to describe them, is not as rich as the actions themselves. However, one must consider the fact that there is little ambiguity as to the meaning of a given word when this word is used in context. To get back to the previous example, we can imagine a machine that could accept the command "send" and which would be able to interpret it at the appropriate level, by asking itself questions like: does something exist which has recently been typed and not sent? If it is a letter, is it signed? Questions like these would allow both the problems of ambiguity to be resolved and the accomplishment of all necessary prerequisites to be verified.

4. CONCLUSION

The different fields of research that have been illustrated may seem to diverge to some extent. However, I am convinced that the differences are very much surface differences, and that the goal we pursue is one and only one. Whatever the domain of research, compatibility is the focus of our work. Compatibility between information display structures and mental representations, compatibility between the concepts evoked by the names of the computer commands and the computer functions, compatibility between the structure of the operator's planning activity and the structure of the software. This is, for us, a key issue for the development of adapted computer systems.

REFERENCES

Bisseret, A. (1983). Psychology for man computer cooperation in knowledge processing. Information Processing 83, LEA Mason (Ed.) Elsevier Sciences Publishers B.V. (North Holland), Amsterdam.

Bisseret, A., Boutin, P., Michard, A. (1979). Introductory elements to ergonomics research in man-machine systems. New Trends in man-machine communication, IRIA, 13-32.

Bisseret, A. Girard Y. (1973). Le traitement des informations par le contrôleur du trafic aérien: une description globale des raisonnements. IRIA Report R37.

Detienne, F. (1984). Analyse exploratoire de l'activité de compréhension des programmes informatiques. Proc. AFCET Conf. on "Approches quantitatives en génie logiciel", Sophia-Antipolis.

Escarabajal, M.C. (1982). Manuel d'utilisation et critiques d'EDIGRA (Editeur graphique). INRIA Technical Report BUR 3207 R11.

Falzon, P. (1982a). Display structures: compatibility with the operators' mental representation and reasoning processes. In Proceedings 2nd Annual European Conference on Human Decision Making and Manual Control. Bonn.

Falzon, P. (1982b). Les communications verbales en situation de travail: analyse des restrictions du langage naturel. INRIA Technical Report 19.

Falzon, P. (1983a). Vocabulary Restrictions in Operative Languages: Towards guidelines for computer commands in restricted natural language (unpublished document).

Falzon, P. (1983b). Understanding a technical language. A schema-based approach. INRIA Research Report 237.

Falzon, P. (1984). The analysis and understanding of an operative language. Proc. 1st IFIP Conf. on Human Computer Interaction. London.

Gould, J.D., Lewis, C., Becker, C.A. (1976). Writing and following procedural, descriptive, and restricted syntax language instructions. Research Report RC 5943, IBM Watson Research Center, Yorktown Heights.

Lafon-Milon, M.T. (1981). Représentation mentale de la verticalité au cours du diagnostic dans le contrôle aérien. III: Représentation des états futurs. INRIA Report CO 8107 R66.

Ochanine, D. (1981). Recueil d'articles. Proc. of a seminar on "L'image opérative", P. Cazamian (Ed.), Paris I University.

Richard, J.F. (1982). Logique de fonctionnement et logique d'utilisation.INRIA Research Report 202.

Scapin, D.L. (1981) Computer commands in restricted natural language: some aspects of memory and experience. Human Factors, 23, 365-375.

Scapin, D.L. (1982). Computer commands labelled by users versus imposed commands and the effect of structuring rules on recall. Proc. Conf. Human Factors in Computer Systems; Gaithersburg.

Sebillotte, S. (1983a). Représentation des actions de l'opérateur. Etude de tâches administratives. INRIA Research Report 256.

Sebillotte, S. (1983b). Analyse prèliminaire du travail de secrètariat dans un service hospitalier. INRIA Technical Report 30.

Senach, B. (1983). Computer aided problem solving with graphical display of information. Psychology of Computer Use, T.R.G. Green, S.J. Payne and G.C. van der Veer (Eds). Academic Press. London.

ORGANISATIONS AND SYSTEMS

NEW TECHNOLOGY: CHOICE, CONTROL AND SKILLS

Chris Clegg, Nigel Kemp and Toby Wall

Social and Applied Psychology Unit
University of Sheffield
UK

This paper examines some of the psychological and organizational aspects of computer-based technology, with particular focus on the use of Computerised Numerical Control (CNC) machine tools in manufacturing engineering. The objective is to introduce and develop some ideas from the fields of occupational psychology and organizational behaviour in ways that will promote an understanding of the uses and impact of advanced computerised technology.

The paper argues against the technological determinist view that once an organization has chosen its technology, then this inevitably leads to a particular form of organization and style of management. The aim is to demonstrate the reverse, that organizations have a choice in how to organize for and manage new technology and that one very important aspect of this choice concerns who has day-to-day operational control of the equipment. The argument is that such choices need analysing in their organizational context since they are in part dependent upon other factors in the organization. Furthermore these choices have major implications for the profile and distribution of skills required in the organization and, at the same time, have a major bearing on the pattern of economic and social benefits and costs which accrue.

The paper draws on material from two case studies undertaken by the authors and refers to relevant theoretical literatures. Before presenting the case material we describe what is involved in CNC machine tool working: first however we briefly outline our research interests in this area and our normal method of working.

1. SOCIAL AND APPLIED PSYCHOLOGY UNIT

The Social and Applied Psychology Unit (SAPU) is attached to the University of Sheffield and is jointly financed by the Medical Research Council and the Economic and Social Research Council of the United Kingdom. Most of the Unit's work involves research into the psychological well-being and effectiveness of people at work. One important aspect of this involves investigation of the psychological and organizational aspects of new computer-based technology, in this instance with particular reference to manufacturing firms. It is our general belief that whilst more and more money is becoming available in Europe and elsewhere to undertake research and development work on the technical aspects of such innovations, little is known about the human side of these changes. Our group is undertaking research and development work into these human aspects because they are psychologically important in their own right, and also because there is evidence that these factors are crucial co-determinants of the overall success of any advanced technical system.

Our normal method of working involves long-term investigations of technical innovation within manufacturing companies. As such we are able to monitor and evaluate what happens before, during and after significant changes. We act as independent researchers accepting no fees for our work and giving each of the significant interest groups in a firm rights of veto over our presence.

Confidentiality of individual views and anonymity for the collaborating organizations are guaranteed, and our research reports are made freely available to all employees within the organization. We do however reserve the right to publish our findings.

We now describe the particular focus of this paper, namely CNC machine tool working.

2. MACHINE TOOLS AND CNC WORKING

Machine tools (be they manual or computerised) are simply mechanically controlled tools for shaping metal - for example drilling holes, making cuts or planing off surfaces. Operating a manual machine tool involves several activities. First, the machine must be set up with the right tools that will drill the correct sized holes or make cuts to the required dimensions (i.e. 'tool setting'). Second, the machine needs setting up such that the material to be drilled or cut is fixed in exactly the right place (i.e. 'machine setting'). And thirdly, the machine needs operating, for example at speeds appropriate for the tool and the material, whilst it performs the drilling or cutting (i.e. 'machine operating').

A computerised machine tool involves the same three activities as a manual one except that, in addition, it needs programming. It is the program (on tape) which controls what the machine does. For example a program can instruct the machine to drill 10 holes in a flat piece of metal using 10 different sets of (x, y) coordinates. Similarly a program can control the movement of the tool in 3 dimensions (x, y, z) at the same time as the movement of the material in 2 dimensions (a, b). This is called a 5 axis program.

As may be visualized some of these programs can be very long and very complex so that after initial preparation they need validating or 'proving'. Usually this is done in two ways. First the machine, tool and tape are run step by step on some cheap practice material such as hardened foam. And secondly, they are run on the real material very slowly step by step under carefully monitored and inspected conditions.

With complex engineering parts in particular, the validation of a tape usually reveals that some editing is required. This is for two reasons, first that there may be genuine mistakes or errors in the program, and second that 'improvements' may well be possible. For example it may be possible to erase some unnecessary operations or alternatively to improve the links between series of tool movements. (These improvements are equivalent to applying work study techniques to the tool in its relationship to the material). Thus in addition to the common activities of tool setting, machine setting and machine operating, CNC working also incorporates programming, proving and editing work. Some physical, manipulative skills however are no longer needed when using CNC machines. The significance of these activities will be considered later.

The advantages of CNC in comparison with manual machine tool working are fourfold. First, some CNC machines can do in a single operation what previously may have required several separate operations on different machines. Secondly, a CNC machine may be able to do it quicker. Thirdly it works with excellent 'repeatability' since the machine and the tape do not get tired or bored. And fourthly it can do some work that would have been almost impossible (or at least exhorbitantly expensive) on manual machines. Technically the advantages of CNC working are such that in some markets in the world, only firms with CNC working are allowed to tender to make some high precision components. It can be seen that CNC machine tools are particularly suited to making small batches of complex engineering parts which are repeated periodically. We now describe the use of CNC

machine tools in two organizations that fit this specification.

3. PRECISION ENGINEERS LTD. (PEL)

This company is a small, non-unionised, precision engineering firm which is part of a large multi-national corporation. The company is located in the UK and employs approximately 150 people. The company operates as a contract jobbing shop making very complex parts in small batches for firms in the aerospace business and in other specialised markets. Almost all the work involves precision engineering usually in small batches and sometimes as 'one-offs'. Many of the orders are repeated periodically.

Over the past 8-10 years PEL has invested heavily in CNC equipment to the extent that, at present, nearly 40 machines are computer controlled in comparison with around 100 manual machines. The latest CNC machines in particular are large machining centres and represent major capital investments of up to £0.80m. For financial and operational reasons these machines are loaded very heavily the aim being to achieve continuous utilization.

Commercially PEL puts in tenders for contracts quoting a price and specifying a delivery date. If accepted the company usually receives the necessary raw materials and a set of drawings from the customer. On receipt of an order a batch is processed by the Planning department. Basically the planners draft a route card which specifies and orders the required machine operations. In addition if, as is typical, some of the operations are to be undertaken on CNC machines, the planners put in a requisition for the Programming department to prepare the necessary tapes.

Production Control receive a copy of the route card and use it to slot the batch into the production schedule booking it in machine by machine working back from the required delivery date. The week prior to beginning work on this batch, Production Control place it on the schedule for the relevant Production Supervisor. By this time the raw materials and drawings (from the customer), the route cards (from the planners) and the program tapes (from the programmers) are available to the Production Supervisor. He simply allocates a man to the first machining operation. On completion the operator marks off that he has completed his operation, the work is checked by a quality inspector and the batch is passed on by the supervisor to the next machining operation.

On average 20 distinct machine operations are required for each batch and the lead time from receipt of an order to delivery of finished goods is around 3 months. At any one time approximately 200 batches are in progress through the system.

Within PEL, management has chosen to staff these machines with skilled men and to give them day-to-day operational control of both the manual and CNC machine tools. Thus on all the machines the operators do the bulk of their own tool setting and always set up their own machines. On the CNCs the men also prove out new tapes, undertake some straightforward programming (not 5 axis) and edit their own tapes to improve the working methods. In the case of 5 axis programs they work alongside the programmer who comes onto the shopfloor to validate and edit these tapes, the emphasis being very much on collaboration between programmer and operator.

The CNC operators report that although they no longer need some physical, manipulative skills they need all the basic engineering skills that they did when operating manual machines , for example concerned with reading drawings and having knowledge of materials to vary machine speeds. In addition they have acquired and

developed some new computer-related skills. They take an obvious pride in their work and in their skills and their commitment to high quality engineering is evident. They report that they enjoy the challenge of making complex parts and are keen to demonstrate to the planners and programmers that they can improve the working methods. They see the task as worthwhile and report satisfaction in completing a whole identifiable component. It is also clear that their operational control of the production process is a source of great satisfaction to them and something they are anxious to protect.

Management recognise and reinforce this high level of skill and the ethos of self-control. For example, the Production Manager said "we've got good blokes here - they're very skilled" and on being asked if he specified machine speeds and feeds a Planner retorted, "we wouldn't insult their intelligence". It is widely accepted that problems arise, are recognised and are put right on the shopfloor and that people there need to have the levels of skill and the degree of control to take the necessary action. The view is that paying 50 pence per hour more for a highly skilled man is a marginal cost bearing in mind the value of the part he is machining (which can be around £4-6,000) and the cost of the equipment he is using (up to £0.80m).

Of course there are both benefits and costs associated with this approach. On the benefits side, the men are clearly highly motivated and committed to good quality work. Whilst they are 'expensive' there is a saving in indirect costs. For example, with this level of problem-solving on the shopfloor, fewer planners and fewer programmers are required than would otherwise be the case. On the other hand however there is no doubt that these skilled men are a very powerful largely self-controlling group. From a management point of view there is too little direct control of the production process. One of the particular problems the managers have is in retrieving information that is available on the shopfloor, for example when an operator has developed and used a good piece of editing. To some extent the operators regard such edits as their private property and as a personal mark. Clearly from the firm's perspective such information should be freely available for general use subsequently. The firm is currently trying to develop a computerised information system which will retrieve such modifications and at the same time give much better data on machine and individual performance. Management here are wrestling with how to keep the psychological and motivational benefits of self-control at the same time as improving management control. One can anticipate there may well be problems with this.

4. SPECIALISED PRODUCTS LTD. (SPL)

This unionised company works in the same markets, makes similar products and uses equivalent CNC technology in comparison with PEL. The major difference between the two companies concerns their size - SPL is very much larger. Our interest here is in how SPL organizes for and manages CNC working.

The short answer is that CNC working in SPL is organized very differently. The machine operators have very little operational control of the machines, their major function being to monitor and mind the equipment. For example the operators are expected to watch the tool to ensure it does not break. Periodically they are also required to press a button on the machine to keep it running - this is to "keep them involved in their work". Real operational control of the technology rests elsewhere in separate highly specialised groups. Thus there is a group of tool setters and a group of machine setters. Their areas of responsibility (and lines of demarcation) are clearly specified. There is also a Programming department which maintains total control of the tapes. Thus programmers prepare, validate and edit tapes and the operators are not allowed to alter them in any way. Indeed, the tapes are locked into the machines so that the operators cannot

handle them.

This lack of control for the operator has both operational and psychological consequences. For example tools wear with use and are expensive to keep replacing. In PEL the operators use their skills to correct for different degrees of tool wear - this is within their control and is another factor for a skilled man to consider. In SPL this problem is resolved differently - here the programmers prepare a set of different tapes for differentially worn tools. Thus there may be 3 tapes for the same job depending on whether new, medium-worn or well-worn tools are used. The consequence for the operator of this set of choices is relative deskilling. The operator does not need machine setting, tool setting, programming, proving or editing skills. Basically he is a machine minder with few decisions to make and little responsibility. In our experience such job designs are usually accompanied by low levels of motivation, commitment and satisfaction. This represents a set of psychological costs for the operators. It can also result in operational costs for the firm if the work system is not 'people-proof' - in other words if high levels of commitment are required, for example, to maintain quality.

The assumptions underlying this set of choices are clearly quite different to those in the previous case. In this instance paying 50 pence per hour extra for direct labour is unnecessary, especially as it escalates with the shift premiums which are necessary to achieve round-the-clock utilization. This money is better spent on white collar expertise in planning and in programming. In this view specialization and expertise are sought, and control is where it should be - with management.

It is also clear that the profile of benefits and costs in SPL is different to that in PEL. For example in this case, direct costs on the shopfloor are lower and direct managerial control of the production process is much tighter. But it is not clear how far indirect specialist groups can go in catering for all contingencies. Thus, is the system effective? One would also predict that the long-term impact of this method of work organization is to reduce the commitment and motivation of the operators. The implicit assumption here is that this can be resolved by extrinsic motivation (carrot and/or stick) whilst, at the same time, the specialist groups (such as programmers) progressively work to reduce the need for any operator involvement. Our personal predictions however are that carrots and sticks are not enough to maintain commitment to quality, and that indirect specialist groups will have difficulty catering for all contingencies. Thus we would expect there to be some operational costs to this strategy.

5. ANALYSIS

The two firms described above are each ideally placed to capitalise on the benefits of CNC working. Thus they each make small batches of high precision engineering parts often in repeat orders. In terms of their markets, products and technology they are very similar. However they have made very different social choices in how to organize for and manage their CNC operations. The evidence of these two cases runs directly counter to the technologically determinist view that the social organization of such innovations follows directly from its technical specification. Our position rejects technological determinism - rather we believe that investment in certain sorts of equipment (such as CNC machine tools) may constrain the range of social choices (see Bessant, 1983), but that significant choices remain. The two firms described above have made very different strategic choices (see Child, 1972) over how to organize for and manage CNC working - below we explore some of their consequences.

In PEL, a small, relatively organically run firm (see Burns and Stalker, 1961), management has chosen to make a "relative investment" in direct employees, giving them operational control of the technology and encouraging them to develop the skills necessary to run it. Accordingly the indirect functions such as planning and programming are organized and perceived as support services to the production process. They support it, but do not control it. Their skills are perceived as complementary to those on the shopfloor and "on call" should specialist help be required. Underlying this choice is a set of "techno-social logics" in which people on the ground are assumed to have the best knowledge, experience and expertise to control the operation. This is directly equivalent to the socio-technical perspective that variances are best handled at source (see, for example, Miller and Rice, 1967; and Taylor, 1978). Clegg (1984) has argued that such assumptions are particularly relevant to production environments where high uncertainty exists and where large amounts of information processing are required, as is the case with small batch, high precision engineering.

This logic stresses the marginality of direct costs and recognises the potential savings that can be made in indirect costs. Adherents of this perspective are sceptical of the benefits of giving control to separate specialist groups since such a strategy raises the problem of integrating their efforts. Failures here can result in such groups setting and following their own objectives and priorities to the overall detriment of the production process. This perspective also runs counter to Frederick Taylor's (1911) Principles of Scientific Management, which stress the need for separating the doing of a job from its planning and controlling.

In SPL, a large bureaucratic and relatively mechanistically run organization (see Burns and Stalker, 1961), management has chosen to make a "relative investment" in the specialist, indirect functions, taking away all the operational control from the shopfloor employees. These specialist groups are not supports to the production process: they are controllers of it. Nor are their skills complementary to those of the machinists - rather they have supplanted them. The skills which in PEL were located on the shopfloor have migrated in SPL to other functions.

The "techno-social logics" underlying this approach lay stress on the benefits of specialization, expertise and direct control. Adherents to this view accept Taylor's (1911) dictum that doing should be separated from planning and controlling, and would suscribe to McGregor's (1960) model of man using Theory X, in which the shopfloor employee is seen as intrinsically unreliable and in need of external control. Of course one of the aspects of this approach is that it can be self-fulfilling. In our experience groups of operators subjected to high levels of external control often exhibit low levels of motivation, which justifies and reinforces the need for further external control.

A further conclusion we draw from these two cases is that the reasons for these different strategic choices over how to organize for CNC working, are best understood in terms of the organizational context within which such decisions are made. Thus PEL is a small, informally managed company with high levels of trust and relatively unsophisticated systems. On the other hand SPL is a large bureaucratically organised firm with a history of specialization. Our argument is that these choices reflect the prevalent "techno-social logics" or guiding values in the firm, which in turn reflect the firm's history, culture and power structures. Whilst it may, in principle, be feasible for a firm to implement CNC working in a way "out of keeping" with other practices in the company, it is unlikely to happen, and unlikely to survive intact should it be attempted. This recognition of a need for congruence within a firm represents a form of organizational/cultural determinism.

Our last point in this context is that these choices result in different patterns of benefits and costs. For example the strategy at PEL has the benefit of high shopfloor motivation through 'enriched' jobs, but the cost of remote managerial control. On the other hand, the choices at SPL entail the benefits of low direct labour costs with the potential costs of poor integration across functions and the risk of relatively high rates of direct labour turnover due to job deskilling. In our experience firms rarely draft up and analyse the pattern of benefits and costs arising from their choices - and, just as significantly perhaps, even less often analyse a different pattern that might accrue from an alternative set of choices. Clearly there are some difficulties with attempting such analyses.

6. IMPLICATIONS

Arising from the analysis above we derive six implications which we believe are of more general interest to our understanding of the uses and impact of advanced computerised technology.

First, the idea of technological determinism is misguided. A particular technology many constrain social choices, but it will not determine them.

Secondly, there is clearly no single homogeneous effect of new technology on operators and their jobs. For example, in one instance the use of CNC machine tools may, for shopfloor workers, promote commitment, the development and exercise of complex skills, and high levels of well-being, whilst in another, the opposite may be the case.

Thirdly, the same argument applies to wider organizational issues. Again the way CNC working is organized in one firm may promote integration across different functional specialisms, whilst elsewhere the reverse may occur.

Fourthly, predicting which sorts of choices are taken, requires an understanding of the organizational context within which the CNCs are installed. We believe that relative investments in the pattern of control of CNCs depend on the logics applied in the firm. These will not be unique to the installation of CNC working but will represent guiding values in the organization reflecting its history, culture and power structure.

Fifthly, it is clear that general ideas of deskilling (by a machine) or even of skill polarization (between experts and non-experts) are over-simplified. In practice what appears to happen is that control over different aspects of the technology and the skills required to control and operate it, are distributed not just up and down the organization (for example between 'management' and 'workers') but also across different functional groups. Furthermore the profiles of these distributions will probably vary firm by firm. One implication of this view is that we need to adopt a more systemic view of control and skill in organizations to capture the subtleties of these profiles.

And finally, we conclude that these different distributions will have different profiles of benefits and costs in organizations and that these may be very difficult to evaluate. For example one organization may need to compare benefits such as commitment and motivation with costs such as low managerial control, whilst another may be considering the benefits of low direct labour costs in comparison with costs such as high rates of absence and employee turnover.

From a research point of view these issues now require careful and detailed empirical study. Whilst of theoretical interest in their own right, they may also help organizations in making informed choices of how to organize for and manage new technology.

REFERENCES

Bessant, J. (1983). Management and manufacturing innovation: the case of information technology. In G. Winch (Ed.), Information Technology in Manufacturing Processes. Rossendale, London.

Burns, T. and Stalker, G.M. (1961). The Management of Innovation. Tavistock, London.

Child, J. (1972). Organizational structure, environment and performance: The role of strategic choice. Sociology, 6, 1-22.

Clegg, C.W. (1984). The derivation of job designs. Journal of Occupational Behaviour, 5, 131-146.

McGregor, D. (1960). The Human Side of Enterprise. McGraw-Hill, New York.

Miller, E.J. and Rice, A.K. (1967). Systems of Organisation. Tavistock, London.

Taylor, F.W. (1911). The Principles of Scientific Management. Harper, New York.

Taylor, J.C. (1978). The socio-technical approach to work design. In K. Legge and E. Mumford (Eds.), Designing Organizations for Satisfaction and Efficiency. Gower, London.

SEMIOTICS AND INFORMATICS: THE IMPACT OF EDP-BASED SYSTEMS UPON THE PROFESSIONAL LANGUAGE OF NURSES

Lars Mathiassen* and Peter Bøgh Andersen+

*Computer Science Department
+Department of the Integrated Study of Computer Science and the Humanities, Aarhus University, Aarhus, Denmark

The aim of this paper is to stress that the use of computers may entail very radical changes indeed in the professional languages used in the affected parts of an organisation. A consequence of this is that analyses of professional languages may be applied advantageously in connection with system development. This paper builds its arguments around a single example: The change in the professional language of nurses in connection with the use of computers in a hospital ward. In the first part of the paper we give a description of the situation before and after the system concerned was introduced. Furthermore we present some basic concepts of semiotics. The second part of the paper contains a semiotic analysis of the situation within the hospital ward under examination. And finally, the last part of the paper draws some general conclusions from the analysis.

1. INTRODUCTION

1.1. The Exeter Case

Before the introduction of the computer system, the situation could be illustrated as shown in Fig. 1. It shows the nurses' conception of their work situation and is taken from the "Edp handbook for Nurses". The figure gives a rough picture of the communication situation of the nurse. At this point the figure serves as a good introduction to the example. We see the nurse in the ward in the center, and around her we see various other groups of people and media with whom or through which she communicates.

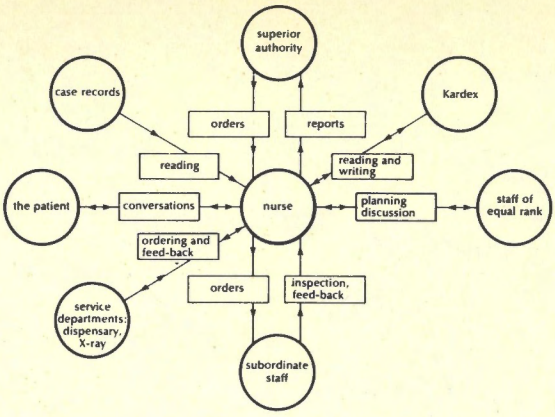


Figure 1 The nurses' working situation before automation, as conceived by the nurses

Here we will focus on her use of the Kardex system. In the Kardex she will enter details on the patients' condition and on prescriptions which have been issued. She will also find information which was recorded on previous shifts. In its physical form the Kardex is a box containing a number of cards; one for each patient. Each card is preprinted with certain columns, and, depending on what the nurse wishes to report about the patient, she writes it in hand in the proper column.

Now a computer system is introduced to replace the manual Kardex, and at the same time part of the information from the case record is transferred to the computer. In the same rough description as we used before, we can illustrate the new communication situation of the nurse as shown in Fig. 2. On first sight the change seems to be of little importance: Earlier the nurse read about the patient on a card from the Kardex box, and, by the same token, she entered information about the patient on these cards. Today she does the same, only now she uses a terminal. But is the situation really the same?

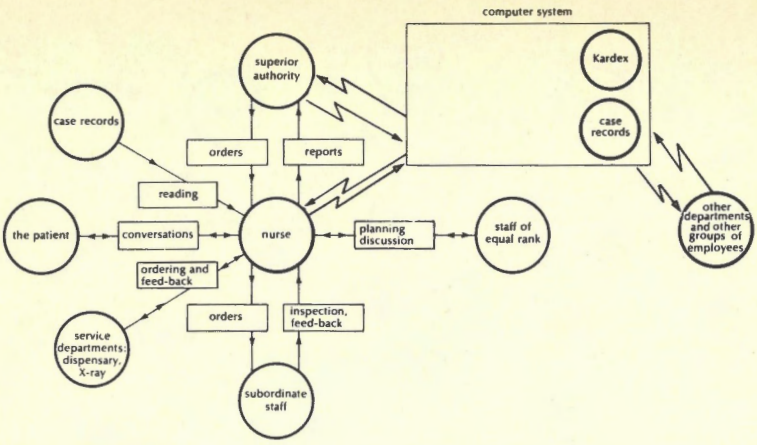


Figure 2 The nurses' working situation after automation

Let us take a closer look at a specific sequence of events after automation. In Fig. 3 we see a display showing a list of the patients in the ward on October 1976 - the nurse will use this in order to select the sub-system which she will use for a given patient.

```
'Monday----25.10.76                    AVON                    NURSING SYSTEMS
     1 MISS ADA MERCHANT        14 MRS AUDREY FARMER        30 MRS ALYS CHANDLER
    11 MRS ANDREA OSTLER         5 MRS AGNES MILLINER       28 MRS ANASTASIA BAKER
SINGLE ROOMS ----------------------------------------------------------------
    29 MISS ALEXIA HAWKER        ------------------         18 MRS ANN COOPER
    12 MRS ANNE-MARIE COOK       3 MISS ANNABEL GROCER      17 MRS AMANDA POTTER
BAY G-----------------------------------------------------------------------
     ------------------          10 MRS ABIGAIL COLLIER      4 MISS ALISON SADDLER
     2 MISS ATHENE SAWYER        19 MISS AIDA CARPENTER     26 MISS AURORA PAINTER
BAY H-----------------------------------------------------------------------
    22 MRS ALBERTA SKINNER       15 MRS ANNIE FORRESTER     16 MRS ALICE PRINTER
     9 MRS AGATHA TAYLOR         25 MRS AUGUSTA BUTLER      20 MISS ANTONIA MILLER
BAY J-----------------------------------------------------------------------
     8 MISS ANGELA FOWLER        24 MRS ANNA BUTCHER         6 MISS AIME GARDNER
     7 MISS ATALANTA SMITH       27 MRS ANTHEA SHEPHERD     23 MRS ALINEA LAWYER
BAY K-----------------------------------------------------------------------


Indicate by O,R,A,D or P the order in which Orders,Reports,Admission,Discharge
or Position in Ward is required:-'
If Orders,or Reports is required then type the numbers of the patients in the
order required:-'

If printing is required, indicate here'
```

Figure 3 Display of the patients in the ward

The figures which precede the patients' names are the internal departmental numbers of the patients. Since it is the order subsystem she wishes to use, the nurse writes an "O" in the space provided on the first line from the bottom. The space between

the two apostrophes is used for input data. Since the nurse wishes to make changes in Mrs. Cook's order list, she writes "12" in the second space. The table shown on the display tells her that she must write "12". When instructed to do so by the nurse, the computer will display the image shown in Fig. 4.

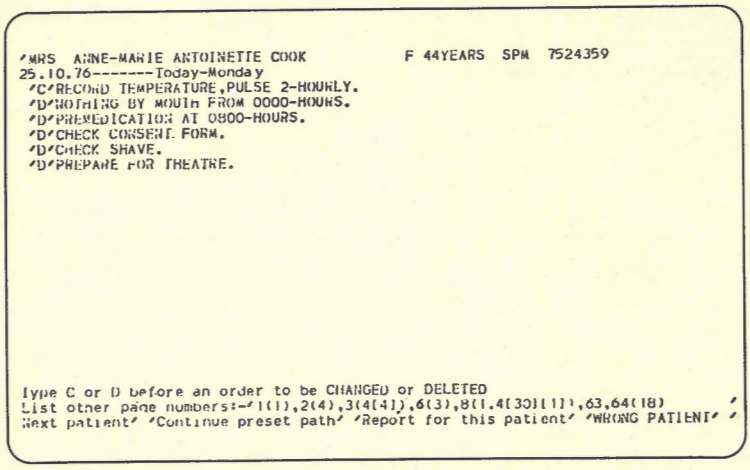


Figure 4 Display of Mrs. Cook's present orders

This display shows Mrs. Cook's present orders. The nurse types a "C" (Change) in front of the orders which are to be changed, and a "D" (Delete) in front of the orders which are to be deleted. The new orders which are to be added belong to the order areas numbered 1, 2, 3, 6, 8, 63, and 64, which are indicated by the nurse at the foot of the page. Where the majority of these areas are concerned, the nurse has also indicated, in brackets, what orders are involved within those areas. The computer is capable of making these changes at once. If the nurse is unable to remember the numbers of the individual orders within an order area, she needs only indicate the area number as in the case of 63 above.

The computer will reply by displaying an image (not shown here) on which the nurse may change the order "Record temperature, pulse 2-hourly" to "Record temperature, pulse 6-hourly". The computer will then reply by displaying an image (not shown here; see Fig. 10) which corresponds to area 63: "Drains". Here the nurse has the opportunity to indicate the desired order within that area. On the next screen image (not shown here), the nurse indicates the desired starting time for each order, and if any individual orders are incorrect. Finally, the image in Fig. 5 will appear on the screen.

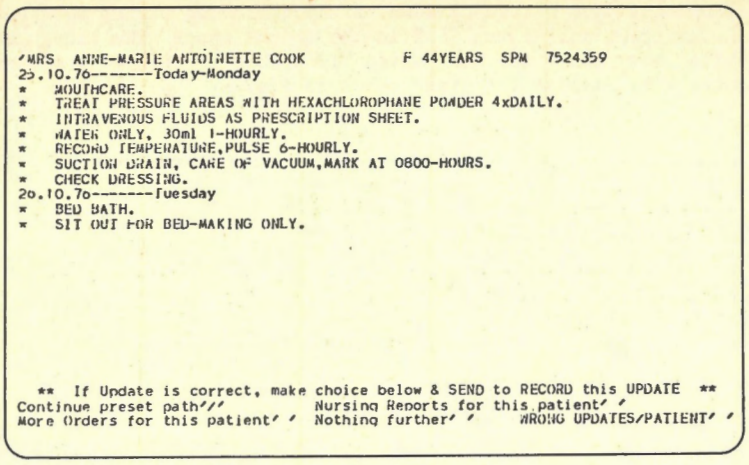


Figure 5 Display of Mrs. Cook's new order list

This image shows the new order list. The new orders are marked with an asterisk (there are no old orders in this example). The nurse can check whether the orders are correct, indicating her acceptance or rejection. In the latter case nothing has happened, and she may start all over again. As a general rule, the new orders for Mrs. Cook will now replace the old ones in the automated nursing report.

Perhaps this may seem overwhelming at first sight; it is nevertheless our view that the system has been well adapted, so that it is not difficult or troublesome to use from a technical point of view.

The computer system has been put into operation with the active involvement of the nurses. Together with computer specialists, a group of nurses has analysed the professional language which is used in the manual Kardex. They have particularly analysed the orders which have been applied. On this background they then, as can be seen from the example above, constructed standard categories and sub-categories of orders. In the new computer based Kardex system there is a total of 64 categories of orders, and it is possible to make additional indications in so-called free text.

These are some basic facts. The second part of the paper offers an interpretation of these facts from a specific point of view: how did the professional language within the Exeter wards change when the system was introduced in the mid-seventies? But first we want to introduce some basic concepts of semiotics.

1.2. Basic concepts of semiotics[1]

A theory of semiotics must treat two main subjects: language usage and language rules [2]. Language rules are rules concerning the interpretation and form of linguistic signs, and they must be shared and supported by a social network of language users. Language usage is the way in which the language users actually apply the rules in communication. By the term semiotic or linguistic community we denote the totality of language rules, language usage, and language users. It is the semiotic community which is the subject of our paper.

We use the word "rules" in its social meaning, that is: rules prescribe how things are done in a correct way from the point of view of the dominant group in the linguistic community. This has two implications: First, that rules can be violated, since they do not represent natural laws. Therefore language usage will fall into two ill-defined subjects: norm-abiding usage, and deviating usage. Second, that deviating usage will in fact occur, since the norm may not be suitable for other groups than the dominant one. Deviating usage is an important move in the continuous struggle about control over language. In particular, it is an important factor in the way in which a language develops.

We may set up two broad categories of language function: the survival function and the development function. The survival function covers the routine usages of language to meet the daily needs, its reason for being. The survival function is normally norm-abiding. The development function may be explicitly marked, as is the case in arguments about linguistic norms, and it may be implicit, as when linguistic groups develop special features which serve to set them apart from the rest of the speech community (e.g., teenagers vs. adults, rural vs. urban speakers, working class vs. middle class, etc.).

The language rules specify how the signs are built. Signs can be described along two dimensions. In the first dimension we distinguish between the content and expression levels. The expression level is that aspect of the sign that serves as a vehicle for conveying the other component, the content level. The relation between content and expression is called the sign relation. Since signs come in many sizes (morphemes, words, sentences, paragraphs, chapters, etc.), there will be many types of sign relation descriptions. Dictionaries describe the standard sign relations on the level of words: the catch word represents the expression level, and the definition is one way of representing the content level. In linguistic varieties of semiotic communities the sign relation is predominantly arbitrary and conventional. There is no inherent reason why a horse should be denoted by the expression "HORSE" and not "COW". Traffic sign posts, however, form another type of sign where the sign relation is non-arbitrary and iconic.

The linguistic rules also describe the combinatorial patterns of the signs, where such exist. There are rules which govern the permissible combinations of phonemes in a syllable, the permissible combinations of morphemes in a word, and the permissible combinations of words in a sentence.

In informatics the distinction between information and data corresponds to the content/expression division.

Two points should be noted:
- the sign relation is a strictly dialectic relation
- the computer is only sensitive to expression level units.

The first point means that expression units and content units cannot exist independently. One level must always be analysed with reference to the other level. The basic analytical method in semiotic analysis is the commutation test. It asserts that in order for two units on one level to count as distinct units, replacement of one by the other must produce a perceptible difference on the other level. For instance, /i/ and /e/ are different phonemes in Danish, since there are two signs, /mit/ and /met/, which have different content, and which differ only in one having /i/ where the other has /e/. Similarly, /male/ and /female/ are different content units, since there are two signs, /han/ ("he") and /hun/ ("she") which have different expressions, and which differ only in one including /male/ in its content, where the other one includes /female/. Also, there is no mechanical way to calculate the contents of larger signs from the contents of their constituent signs. New content units, new patterns of distinctions may come into existence.

The second point relates to the use of computers. A computer can manipulate strings of characters, but there is no evidence that its mechanical processes bear any deeper resemblance to the feats humans accomplish when they use language. It may

mimic human behaviour during a short interval, but the deception is revealed sooner or later. The activity of program design and implementation should be seen as an activity aiming at creating means of expression. Technical descriptions of computer systems play a role similar to phonological descriptions in semiotic theory: they provide structure for the expression level of the signs. In fact, abstract data structures and algorithms correspond to phonemic descriptions, while implementations of these items correspond to phonetics.

We can also describe signs along another dimension, characterised by the opposition substance versus form. The general idea is the following: Imagine some continuous part of reality, e.g., the colour spectrum. What language does, is to articulate this continuum into parts that the language user in some sense considers different from each other. Thus one part of the colour continuum is considered "red", another part is considered "blue", etc. The colour continuum is an example of what we generally denote the purport of the sign. The purport thus refers to that part of reality which we describe by a given sign. The term substance then denotes the articulation of the purport by some form, whereas the form is the abstract principles of organisation, defining types of signs.

Finally, if we combine the two dimensions of signs, and consider the content form in its totality, it seems to fall into minor partitions, each partition being characterised by an especially tightly knit network of oppositions of relations. Such a partition of the content form is called a semantic field. We have already met one: the field of colour units. Other fields which have been investigated include: kinship relations, verbs of motion, verbs of communication (speech-acts), verbs of possession, etc. Especially in historical linguistics, the concepts of semantic fields has proved useful in describing semantic changes.

The concept of form is a structural concept. Here structure means a network of units which mutually delimit each other by differences and oppositions, somehow holding each other in check in some kind of equilibrium. This structure is not fixed. It develops historically and is constantly under attack from different groups in the speech community, each group trying to design the structure according to its interests and needs. It has certainly ideological significance whether the political semantic field consists of an opposition, "communism" vs. "the free world", or of "planned economy" vs. "capitalism".

Fig. 6 shows a more innocent type of semantic fields, the verbs of possession (cf. Bendix, 1966):

	A has B after time T		A has not B after time T
	B is not A's	unmarked with respect to "B is not A's"	
C causes it	C lends A B	C gets A B	
A causes it	A borrows B from C	A takes B from C	A gets rid of B
chance causes it		A finds B	A loses B
		A gets B (unmarked with respect to causation)	

Figure 6 Semantic field of verbs of possession. The distinctive features are: causation, ownership, time (from Bendix, 1966, p.76).

The verbs contrast according to three dimensions: causation (/C causes it/, /A causes it/ or /chance causes it/, ownership (marked or unmarked with respect to /B is A's/), and time (A has or has not B after time T). Notice that the same signs may be used to articulate other types of purport, for instance the continuum of health states ("I've got the flu", "I've lost my health") or emotional states ("I gave her my love").

Fig. 7 shows a description of the spoken version ot the sign "take":

	content	expression
form	/A has B after time T/ (~ get rid of) /A causes it/ (~ find)	t e^{i} k (t ~ p,k,.... e^{i} ~ a^{i}, o^{u}, etc. ...)
substance	articulation of the continuum of possession states and changes	articulation of the sound continuum

Figure 7 An analysis of the spoken sign /t e^{i} k/.

Note how the content and expression forms are defined solely by the way the sign differs from its neighbours.

The form/substance distinction shows its empirical validity in predicting such "multiple-uses" of signs. Sometimes the form of one semantic field is more or less spontaneously transferred to organise a different type of purport, in the beginning

with a clear metaphorical tinge. This has been the case in the area of man-machine interaction which has been modelled over forms of understanding imported from man-man communication.

The important points to be remembered during the remaining parts of this paper are the following:

- a language community exists through an interplay between rules and usage, rules governing the usage, usage complying with, or opposing rules;
- the structure of the content level of language is intimately connected to the cultural and political aspects of the language community;
- changes in the content level of a language occur frequently, and a change in one place may have repercussions throughout the whole structure;
- the content and expression levels are dialectically related, on one hand they are distinct, yet on the other not able to exist separately, like the two sides of a coin;
- the activity of programming aims at designing new expression forms which are realised in the purport of micro-electronic processes and states;
- the survival and development functions of natural language are intimately interwoven.

2. SEMIOTIC ANALYSIS

In this second part of the paper, we shall use semiotic concepts to describe the changes in the Exeter ward. The following points will be made:

(a) The expression level changes: the new expression substance makes it easier to realise the expression form in different substances, and the expression form becomes more restricted (section 2.1.).
(b) Similarly, the content form becomes less fine-grained. However, the borders between content units also change: from being fuzzy and based on personal experience and typical examples, they tend to become sharper and more based on standard definitions (section 2.2).
(c) Also, the sign relation changes: the relationship between the expression and content level tends to become distorted and opaque. It is not clear which speech acts are executed by means of the system (section 2.3.).
(d) Finally, we remark that the new signs of the edp-based system also convey connotations (section 2.4.).

It should be emphasised that the following descriptions are our interpretation of the statements made by the nurses, as documented in the "Edp Handbook for Nurses" chapter 5.

2.1. Expression level changes

Let us take a look at a fraction of the Exeter semiotic community after the new edp-system has been taken into use. The semantic field is time and frequency for application of drains. There are 8 signs, as shown in Fig. 8.

time or frequency of drain application			
frequency		time	
definite	indefinite	relative to clock	relative to nursing work
DAILY 2×DAILY 3×DAILY	WHEN NECESSARY AS DIRECTED	AT ?-HOURS	BEFORE BATH AFTER BATH

Figure 8 The semantic field of time and frequency of drain application

The relevant distinctions seem to include: frequency vs. time, clock time vs. relative time, definite vs. indefinite, and the numbers. Fig. 9 shows a tentative analysis of the sign "DAILY".

form	definite (~ WHEN NECESSARY) frequency (~ AT [?]-HOURS)	/DAILY/ (built of D,A,I,L,Y, all characters in opposition to the other members of the alpha-numeric character set)
sub-stance	time and frequency of drain application	pixels on screen printer output magnetic spots on disks or tape

Figure 9 An analysis of the "computerised" sign /DAILY/

If we compare to preautomation times, we immediately note the differences. For example, before automation the expression substance was paper, and this meant that it was difficult to change it. The computer media vastly increase the possibilities of realising the same expression form in different substances: pixels on the screen, dots or lines on a printer, magnetic spots on discs or tapes. On the other hand, the expression form has been severely restricted: before, the nurse could use handwriting with all its numerous possible distinctions - now she is confined to the distinctions allowed by the standard character set.

The example can readily be generalised: the expression level of the semiotic community is the part which is most directly influenced by automation. Clearly, the expression substance is changed, but normally also the number and structure of possible expressive distinctions change, being limited by the character set offered by the terminal and printer. In the Exeter case, the restriction went even further. As mentioned in section 1.1., a standard list of allowed expression units was compiled and stored in the computer system. Clearly, at least the expression level changes drastically, both with respect to form and substance, when computers are introduced. This observation is also relevant when we try to assess the impact of edp-technology on different professional groups, since the content/expression distinction is reflected in a social division of labour. Journalists, writers and managers are mostly concerned with the content level of the signs they produce, whereas printers, television technicians and clerks to a larger degree work with the expression level. Therefore, the latter groups are most directly affected, for example through higher unemployment rates.

2.2. Content level changes

But what about the content level? After all, this level is the important one, since expressions are only means of conveying the contents. The point to be remembered is that the content structure of semiotic communities cannot help changing when the expression level changes, the reason being that the content units, in order to preserve their social existence, must belong to signs which have different expressions, else they could not be communicated. Thus, if means of expression disappear, their contents also disappear. But since the content units are defined by their mutual differences, disappearance of one content unit will cause the remaining ones to change their boundaries, and, by the same token, their definitions. This again changes the general perspective of the language users, since their conception of reality is to some degree dependent upon the language at their disposal.

Let us look at the Exeter case again, and focus on the same fraction as above. Previously, the part of English that constituted the nurses' professional language allowed them a rich variety of semantic distinctions. As mentioned above, now only 8 remain, or, more correctly, 8 are recommended - it is possible, but inconvenient, to enter free text into the system, and it runs counter to the intention expressed by means of the system. The semantic field of time and frequency is articulated in a much less fine-grained way than before automation. But there is more to it than that.

First, the semantic fields are not fixed in natural language. They change in complex ways, depending on context and point of view. Consider the word red about the colour of hair. In an advertisement for hair dyes, it may cover a vast semantic field with many shades: fair, blond, dark-blond, auburn, brown, black, or what have you. But when I say to a male friend "I like read-haired girls", its neighbours are probably only "fair" and "dark". It is certainly not all the fancy shades conjured up by the hair dye manufacturer in his advertisement. And in literary works, the author can consciously build up semantic fields with even fewer members, e.g. "red" vs. "non-red", loading the distinctions with other distinctions important to the theme of the book. Also in everyday conversations, semantic fields are continually adapted to situation and point of view.

But in the Exeter system, the semantic field is fixed once and for all. "Daily" gets its meaning from the same 7 neighbours in all situations. Period!

Secondly, we can also say something about the way the structure of the semantic field changes. The boundaries between the elements of this structure can have different properties. In our case, the distinction between fuzzy and sharp boundaries in relevant. In everyday language, the boundaries between the elements of a semantic field are fuzzy. There are portions of meaning which do not clearly lie in one section or another. Consider e.g. the word vegetables. Potatoes are clearly vegetables, but what are strawberries?

Concepts are special examples of elements of semantic fields. Not all elements are concepts. For example, it is not clear which concepts are expressed by emotive words like "oops" or "damn", or by prepositions such as "for", "with", "by". By a concept we understand an element of a semantic field which has an extension and intension.

The extension denotes the actual phenomena of the concept, i.e. it denotes part of the purport related to the semantic field. The intension, on the other hand, denoted those properties (distinctive features) which characterise phenomena in the extension of the concept. From what has been said above, we can distinguish between two different idealised types of concepts: the Aristotelian and the fuzzy or prototypical concepts.

In Aristotelian concepts, the intensions consist of a number of properties, which are common for all phenomena in the extension of the concept. Thus, the phenomena in the concepts' extension possess all the properties of the concept's intension. As a consequence of these characterisations, we assume here that concepts can be classified hierarchically by means of division and sub-division of the broadest and most common concepts. Thus, a concept on a higher level in the hierarchy always has a more comprehensive extension than that on a lower level; while a concept on a lower level has a more comprehensive intension. However, the hierarchical structure is not unambiguous, as it is possible to build a number of concept-hierarchies over the same phenomena (e.g., human beings may be sub-divided into children and grown-ups, or into male and female).

As the intension of the concepts is fully defining, the question of membership of a concept's extension can always be settled with yes or no. The Aristotelian concepts are characterised by sharply defined conceptual boundaries.

Where fuzzy or prototypical concepts are concerned, the intension similarly consists of a number of properties. But even though all properties apply to some of the phenomena in the concept's extension, it does not necessarily follow that some properties apply to all phenomena in the extension. Here various phenomena are seen as more or less typical specimen, and the most typical specimen are characterised as prototypes belonging to the concept. Whether a phenomenon is seen a part of the extension of a concept is, among other things, based on the relative resemblance between the phenomenon and the concept's prototype, and these assessments may vary from person to person, depending on experience (Larsen, 1980).

In the following we will argue that the introduction of the computer based Kardex probably entails a shift towards more Aristotelian concepts in the nurses' professional language. If we can substantiate this thesis, we have rendered probable that at least some kinds of automation may cause very fundamental changes in the semiotic community. Until now, we have only shown that the number of content distinctions decreases. We have not shown that the remaining boundaries become more sharply defined; in fact, exactly the opposite could perfectly well happen.

The main reason why the content form will become more Aristotelian must be sought in the area of language usage. We will first make some comments on the development function, and then we will comment on the survival function.

2.2.1. Changes of use: development

In the following we will describe some changes in the mode of development of the Exeter semiotic community. It is a well-known fact that computer based systems tend to change in leaps: We have stable periods where a particular version of the computer system is running on the machine; during this period, users may discover unwanted properties in the system, or the surroundings may change so that aspects of the system become out-dated. But in spite of the growing dissatisfaction, the system remains stable. At some point, the problems become too big, and management decides that the system must be changed. The changes are specified, project groups are established, and the system development process is planned and executed.

Historical linguistics tell a different story about the manner in which semiotic communities change: here, changes occur in intimate connections with language usage (in its survival aspect). Subcultures of language users create innovations, some of them blossom and die, whereas others spread their seeds to other groups of society. In fact, many usages of languages inherently involve language changes: discussions often focus upon the proper use of words, participants try to convince each other that their conception of a word or phrase should be used, or they create new words or expressions to meet needs which are not served by standard expressions. Several of the papers on the present conference, including our own, are deeply engaged in these activities!

Thus language changes and fluctuates all the time. Government agencies have few possibilities of planning a language change, dictionaries are mostly attempts to systematise and regulate changes which have already occurred.

The point is not that language change is some kind of unconscious and natural process; on the contrary, it may be very conscious and political, as for instance the Norwegian language situation indicates. The point is that the relationship between language usage and language change is more intimate than in computer based semiotics. Natural languages may, to a large degree, be used to talk about themselves, they include signs whose content are other signs.

A similar dialectic is not possible in the Exeter system. Here the language rules have been objectified. They exist as data in the computer, and can be shown on the screen. The screen displays a grammar of a subset of nursing orders. The grammar is described in a slot-and-filler formalism, consisting of a few columns, each one containing al the constituents allowed in this position. The nurse selects one constituent from each column to form a complete nursing order. See Fig. 10

```
'MRS  ANNE-MARIE COOK          DRAINS                          Monday----25.10.76
                          :20 OBSERVE                          :
DRAIN  1 ANIRAL           :21 CARE OF VACUUM                   :40 DAILY
       2 CORRUGATED       :22 CLAMP                            :41 2xDAILY
       3 POLYTHENE TUBE   :23 CLEAN & DRESS WITH DRY GAUZE     :42 3xDAILY
       4 RUBBER TUBE      :24 CLEAN & DRESS WITH PARAFFIN GAUZE:43 AT [?]-HOURS
       5 SUCTION          :25 EMPTY DRAINAGE RECEPTACLE        :44 WHEN NECESSARY
       6 T-TUBE           :26 SHORTEN BY [?]cm.                :45 BEFORE BATH
       7 UNDERWATER SEAL  :27 IRRIGATE                         :46 AFTER BATH
8 PARACENTESIS OF ABDOMEN:28 REMOVE                            :47 AS DIRECTED
9 SOUTHEYS TUBES          :29 DOCTOR TO REMOVE                 :
                          :30 ? REMOVE ?                       :
50 PUMP VALVE [?]xDAILY


List additions'5,21,99[MARK AT 0800-HOURS]
                                                        'NO ADDITIONS' '
```

Figure 10 A display of possible orders concerning drains

For instance, the nursing order "suction drain, care of vacuum, daily" is constructed by writing 5, 21, 40. There can be no immediate feed-back from language usage to language rules. Dissatisfaction has to be accumulated before it is technically and economically feasible to change the system. This again means that there must be a gap between the experiences that cause language change and the change itself. The experiences must be analysed and standardised in order to support programming.

Here we have the first factor which favours a shift towards Aristotelian concept formation: one of the reasons why natural language uses prototypes is that it more directly reflects the immediate and practical experiences of the language users when

the language is learned and developed. And one of the reasons why natural language concepts are fuzzy is that concepts change continually and anarchistically. If the concepts used by two language users are compared, they may have changed in slightly different directions. When they talk, and the concepts must function as one and the same concept, it will exhibit fuzzy boundaries, owing to the diversification.

2.2.2. Changes of use: survival

We have pointed out some differences between computer based semiotics and non-computer based semiotics in the mode of development. Now we will focus upon another aspect of usage, the daily usage, the practical reasons for having the system at all: the survival function.

First we will introduce two kinds of knowledge, connected to the prototype/Aristotelian dichotomy mentioned above (Larsen, in press). First we have episodic knowledge. Episodic knowledge is about personal experiences, events with known actors, fixed with respect to time and place. Episodic knowledge is created by the language user himself: it is an interpretation of experiences. On the other hand we have semantic knowledge which is about states of affairs: about concepts, their inter-relations and possible usage. Semantic knowledge includes knowledge about the language. Semantic knowledge is, to a large degree, second-hand knowledge, acquired through reading or instruction. A typical example is a botanist's knowledge of the botanical species and sub-species.

Let us once more take a look at the Exeter system, and consider the situation as it was before the computer system was introduced: A nurse observes a patient; maybe she reads the case sheet; she estimates the patient's medical and nursing needs; and taking everything into account, she writes some orders in the Kardex. Maybe she uses standard terms, maybe she expresses herself in a more varied and personal manner because she estimates that the situation demands it. Later, another nurse reads the orders, and nurses patient accordingly.

After the introduction of the computer system, the situation is different: Again the nurse observes the patient, and maybe she reads the case sheet; however, by using the computer system daily, she has learned the given standard categories and sub-categories, and she will observe the patient and read the case sheet through those glasses; maybe she estimates that the situation requires that she writes the report in free text, but normally she will employ the given orders - because they are there, and because it is easier.

In the former situation, it is certainly possible for the nurse to take pain to interpret a personal experience in the light of previous experiences, and create a meaning which is intimately bound to it. The knowledge recorded may have the characteristics of episodic knowledge. The entry can be loaded with context-dependent meaning, of which a good deal will be interpretable to the other nurses in the ward, owing to their shared background. In this case it is certainly possible to perform the speech acts of interpretation and description.

In the latter situation, things are different, since the system invites a different mode of usage. The groups of nursing orders are classified in a two-level hierarchy, and the nurse must specify the order class before she can enter the order. This means that the general concept to which the "description" belongs, must be specified before the description can be created. A movement from the general to the specific is mandatory, and the speech acts performed will probably tend to be some kind of determination of species, choosing between a fixed set of categories. One can doubt whether episodic knowledge can be recorded in this manner. However this may be, there is no doubt that semantic knowledge is a necessary prerequisite for using the system at all. Thus one may fear that the semiotic community changes, from a situation where a group's personal and collective experiences are couched, interpreted, and developed, to a situation in which a dominant group forces its classification scheme upon less powerful groups.

We have argued that fundamental changes are to be expected when computers are introduced as tools into existing semiotic communities. In particular, we have argued that the Exeter system probably will mean a shift in emphasis, from episodic knowledge, based on prototypical concepts, to semantic knowledge based on Aristotelian concepts.

The conclusion is deliberately formulated in vague terms (e.g. "probably"). A more positive statement would demand extensive empirical studies of the semiotic community before and after the introduction of the Exeter system. On the other hand, it should be pointed out that the nurses' own appraisals in "Edp Handbook for Nurses" support our conclusion:

> It will be possible for important information relating to the patient to be lost, due to the fact that this reporting system does not provide the same opportunities for expression. For instance, it is the duty of the nurse to help a patient who has indicated that he has a problem which he is unable to solve himself. This fact should appear in the report. But what is one to do in the absence of a standard formulation for precisely this problem? Either the problem will not be reported, or it will again become necessary to adopt a dual reporting system. (...) There are many things which may not be standardised. For example, observations, the patient's mental state, and the patient's mood. How could such things be standardised? These matters are so subjective that they must, by necessity, be reported verbally in order to ensure that the recipient of the report has actually understood what is meant in the report. Thus the danger exists that the most significant items may not be included in the written report and may thus slide into the background. It is, in fact, these very items which are most difficult to describe when using standardised terminology.

2.3. Changes in the sign relation

Descriptions of the sign relation in greater detail specify how the content units are expressed by means of expression units. And in this case, where a semiotic community changes through the deliberate introduction of a new edp-system, one of the interesting questions to ask is: How, and to what extent will the new means of expression be able to convey the content intended?

In relation to explicitly designed parts of semiotic communities, we will suggest the following demands to the quality of the sign relation:

> The principle of transparency and efficiency
> - the expression level should be designed in such a way that it clearly indicates the type and effects of the speech act conveyed, and ensures the speech act a reasonable chance of succeeding.

Commonplace as it may sound, the criterion is by no means met by many of the systems with which we are familiar. For instance, it entails severe restrictions upon the structure of integrated information systems, discouraging extensive automatic manufacture of new data sets (Nurminen, no date).

In fact, we shall see that the Exeter system seems to run counter to the principle in one respect - by the way, a point that gives some of the reasons why the expressive system was standardised in the manner described. One of the reasons for automating case sheets and Kardex is to make reporting faster and more reliable. This will enable management to allocate nursing resources on a day-to-day basis, one possibility being to assign the individual ward a basic allocation of personnel, and to maintain a buffer of nurses who are assigned to different wards every day, following the work-load predictions of the system. The net effect from management's point of view is improved economy: the net effect from the nurses' point of view is corruption of their work community.

The nurses themselves give the following appraisal in "Edp Handbook for Nurses":

> If a computer were to be introduced which was similar to the system used in Exeter, then nursing management would have direct access to all the information which had been collected from the departments via the terminals. The staff in the departments would have no influence on what information was used, nor would they even be aware of when it was being used. The nurse's report, which was kept in a Kardex system before, would no longer be a tool for internal use within the department. The management would be able to see from the automated nurse's report what examinations, treatments and nursing care procedures were being provided for each individual patient. This would permit the work load to be calculated on a daily basis. These calculations, when combined with the information relating to patient occupancy levels, the nursing work load, the number of nursing staff, and the number of absences due to illness, would place the management in a much better position to direct the distribution of the nursing staff throughout the various departments.
> In addition to the fixed staffing levels, a pool of nurses would be maintained, who would not be employed in specific departments, but who would be allocated as required. Let us say, for example, that the calculation for the following day shows that a certain department with a fixed staffing level of 4 will in fact require 6.4 nurses, then precisely 2.4 nurses would be allocated to that department form the pool.
> The result of this will be a reduction in nursing care. If the patients are to get used to new staff every day, there will be no continuity of care, and we ourselves will no longer enjoy a secure working environment.

Automatic allocation presupposes that weights are assigned to every nursing operation, and since computers can only handle expression units, it entails a severe standardisation of the nomenclature of nursing orders. Never has history witnessed such a tight connection between economic objectives and their linguistic effects!

The reporting is no longer done by the matron, but by the nurses themselves. And the means of expression have changed radically; before, the reporting was executed by means of expression units which were distinct from those conveying orders and nursing reports. Now, the same expressive actions (inputing nursing orders, and reporting them when they are done) convey two different acts:

(a) those of ordering and reporting nursing to other employees in the ward,
(b) those of reporting work load to management.

Our evaluation criterion is violated since this semantic difference should be marked with different expressions. In fact, the nurse may not even be aware of the fact that she does the latter type of reporting, and in that case the speech act of reporting should be considered defective, since the sender does not intend it. And who is the sender of the ensuing allocation speech act on the content level? It cannot be the computer. The sender must in fact be the management, since they decided the properties of the data which made the computer produce the allocation sheets. Yet on the expression level, management will receive, not send the print-out. Thus, from a semantic point of view, things have been turned upside-down on the expression level. The sign relation between allocation sheets and their contents, allocation orders, has become opaque. Clearly, distortions of this type are not unique to the Exeter ward.

The last point to be made with respect to the sign relation is the following generalisation: The expression level of every computer based semiotics must contain the configuration shown in Fig. 11.

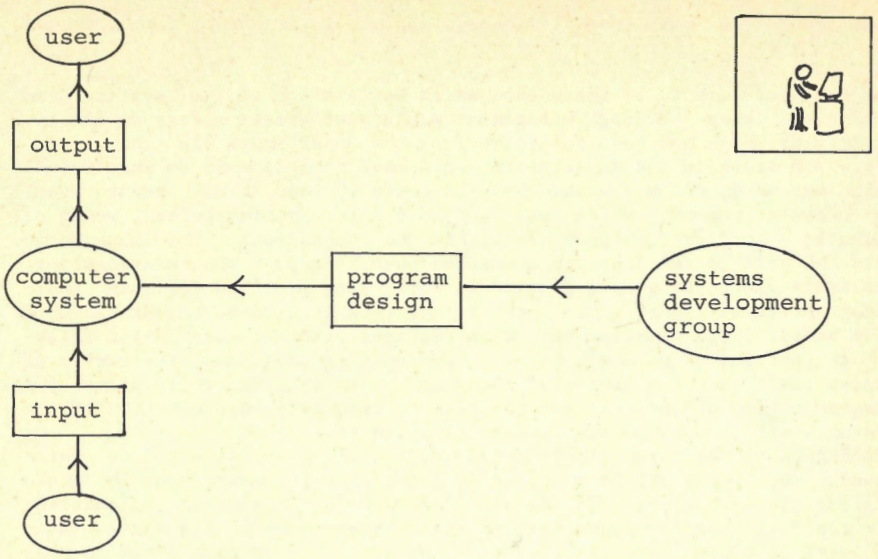


Figure 11 Basic configuration of expression level description

A system development group writes descriptions, some of which are read by the computer. This enables the computer to read and write other texts. On the content level description, the computer must be eliminated as a sender/receiver, and the contents of the program - if any - must be shown as a direct relation between users and systems development group, as shown in Fig. 12.

We have argued that the system implies changes in the semiotic community: from being based upon the speech acts of description and interpretation, it shifted towards classification and determination of species. Who is the sender of this content? Still not the machine, since it is only a medium through which the communication is effected.

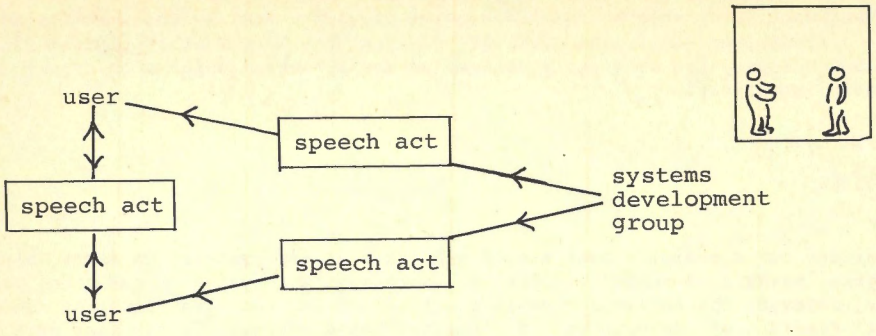


Figure 12 Basic configuration of content level description

There is no escaping the fact: the senders are the participants in the systems development group: management, systems analysts, and nurse representatives.

2.4. Connotation signs

It is worth while to take a closer look at the speech acts of nursing and disease definitions, since they are conveyed by a special type of signs called connotation signs. If you look at the screen displays, you will see that they do not denote general properties of nurse work. They denote possible and actual nursing orders and patient health states.

But signs may enter into other higher-order signs. In particular, a sign denoting properties of reality, may be the expression level of a connotation sign. Thus, the form of a connotation sign is: ((expression, content), content). The formula illustrates the special mode of signifying of connotation signs: they systematically denote their content by denoting something else. Connotation signs are rather effective persuaders, as is attested by advertisements which draw heavily on connotation signs. A soap advertisement denotes a group of young people crowded into a car, but it connotes a quite different content: a better life, fun, a feeling of belonging. And it is the latter argument, not the former, which makes us buy Rexona. For a treatment of connotation signs, see Barthes (1957). Since edp-systems are parts of semiotic communities, it should not come as a surprise that they also convey connotations. There is nothing wrong with that - all semiotic systems do. The point is rather that connotations should be taken seriously, since they are transmitted through the community day in and day out.

The nurses themselves are very conscious about the fact that new concepts about patients, diseases, and nursing may be built into systems like the Exeter system. "Edp Handbook for Nurses":

> The danger exists of the patients simply becoming pawns in the game. The individual patient will not receive individual treatment. (...) It is conceivable that specific health policy will be incorporated in the choice of orders. E.G. socio medicine. In this case there would be both political and central direction of the nurse's work. (...) The nurse will possibly lose a certain amount of interest in her work, since she will be denied a proportion of the opportunity to take the initiative independently. There is less of a challenge in choosing numbers than in formulating sentences.

Do nurses and management - each from their point of view - want to change their conception of patients, diseases, and nursing? How can they know whether the introduction of the new edp-system will imply changes on these levels which correspond to each of their intentions?

3. SUGGESTIONS

We have carried out a semiotic analysis of a given computer system. On first sight the analysis reveals a number of critical aspects in the system in question. More indirectly, however, the analysis reveals shortcomings in the practices or strategies of traditional development of computer based systems. In the last part of this paper we will try to formulate our criticism constructively in the form of alternatives.

3.1. Alternative systems and strategies

Let us start by outlining other ways in which computer systems might be applied in relation to nursing. We do not think that hospital systems of the Exeter type will benefit nurses and patients since a major objective of these systems is to increase external control of the nurses, and decrease costs. It may well be that if these objectives are not acceptable to nurses and patients, they may see no reason at all for using edp-technology.

However, there might be problems on the ward which could be solved easily by means of computers:

- Kardex and case sheets are sometimes needed by several people at the same time;
- nurses work in shifts and it is difficult to draw up time-tables;
- more than one nurse takes care of the same patient, and it may be vital that the nurse knows what has been done on previous shifts.

One may envision a system which is only used locally by the individual ward. The system will accept nursing orders and patient descriptions in free text, and it contains a good editor which makes corrections easy. It can display the Kardex record for the individual patient. It might even be a system which could store and display pictures, e.g. X-rays. Nurses who have just reported on duty can display entries made during the previous shift. The system can also show nursing orders which are to be effected by the nurse who takes over.

As a further facility the system contains edp-based tools for planning. These tools are not meant for automatic planning, but they help the nurse in seeing the consequences of her choice in drawing up time tables. Finally it might be useful to have some kind of concordance or lexicon in the system. A concordance system would show all contexts in which a word or phrase occurred: if a nurse is unsure of how to use a word in a concrete situation, the list will show her how her colleagues have used the word previously. The ward may want to maintain a lexicon of standard definitions. This lexicon should be dynamic, and it should be controlled by the nurses. It would objectify the current consensus among the nurses on use and meaning of important words. Such a system might be useful and avoid a good deal of the criticism levelled against the Exeter case. This system would basically be used as a means of expression and would exploit the computer's remarkable ability to realise the same expression form in different substances and different graphical shapes. The content is produced and reproduced by a closely knit group on the basis of shared education, experience and interest.

The above-mentioned system is technically perfectly feasible. But is it attractive from an economic point of view compared to the Exeter system? Ciborra (1981) maintains that, seen from an economic viewpoint, there are two fundamental strategies

for the application of computer technology in organisations. One seeks to improve the efficiency of the coordination of the working processes through a standardisation of the information systems. The other seeks to improve the efficiency through extension of linkages between the parties involved in the working processes. Real-life practice will always be a combination of the two strategies. The Exeter system, however, leans heavily to the first strategy: standardisation. As opposed to this, our alternative proposal is primarily based on the second strategy. Hence it is not an economic argument as such which sets our proposal apart form the Exeter system. As mentioned earlier the most important difference is that the Exeter system facilitates a close external control. This will not be possible in the alternative proposal.

By applying the sytem we have outlined, the nurses will gain experiences which, through a process of stepwise structuring, may lead to new applications (Sandewall et al., 1982). In this context the Exeter system presents a "ready-to-wear" solution to a management problem, while the system we have outlined permits a process of experimenting based on the interests and experiences of the nurses (Budde et al., 1984).

Choice and evaluation of the two proposals and the two strategies for development depends on the perspective applied. All development processes require descriptions as a basis for realising the system. Traditionally, these descriptions are confined to being expression level descriptions. They are based on what we will call the system perspective. This perspective offers a relatively narrow view on the application of computer technology, and we suggest that it, among other things, is supplemented with the communication perspective.

The system perspective focuses on the expression level. The elements of the expression level form a simple part-whole structure and are amenable to decomposition techniques. The edp-based system consists of several components, e.g. users and edp-system. Signals are transmitted between users and edp-systems, and there is structural analogy between edp-systems and user component. In this perspective it is sensible to talk about man-machine "communication" in one sense of the word. Expression level descriptions in the system perspective can be used as drafts for programs. The theme of these descriptions is how data are transmitted and manipulated by formal rules.

Aspects of the content level may be described in the communication perspective. In this perspective, communication is solely between humans, and the edp-system acts as a channel or medium for this communication. It is a tool for storing and transmitting expression units, i.e. data. Humans and machines must be described by very different types of concepts since they play different roles in the communication process: humans are sensitive to both content and expression, whereas machines are only sensitive to expression features. In this perspective, it makes as little sense to talk about man-machine communication as to talk about man-tv-set communication, or man-paper communication.

From a linguistic point of view we recommend that methods for systems development are extended to include the viewpoints offered by the communication perspective. The communication perspective gives theoretical support to experimental methods in the following sense:

(a) The purpose of many edp-applications is intentional creation of meaning in humans;
(b) Creation of meaning cannot be predicted from expression level descriptions alone, since it depends on the context of the interpretive process (cf. section 2.2.2.). Experience from linguistic and literary analysis indicates that content analysis to a large degree must be post festum.
(c) Therefore, important properties of many edp-systems (i.e. production and reproduction of meanings) cannot be assessed until the system is used in a realistic context by its future users.

Rigid use of phase-oriented methods is comparable to a process of writing teaching material where the exact structure of chapters, paragraphs, sentences, footnotes, and references is determined alone on the teacher's vague ideas on the content of the material. The form of the material is fixed before any practical insight into the actual teaching situation has been obtained. No teacher would even dream of embarking on such an endeavour.

But the communication perspective has its own limitations. Firstly, it must be embedded in descriptions of the larger, possibly noncommunicative, work processes since, once again, the interpretation depends on the context of work and organisation, not to speak of cultural and social contexts.

Secondly, experience shows that difficulties arise in organisations with a sharp functional division between "expression level workers" and "content level workers" (cf. section 2.2.1.). However, these difficulties can be used productively since they may be used to question this division of labour.

Thirdly, there are applications, notably automatic control of machines, where the edp-system is not used for communication, since the receiver of the signals is not a human but a machine. However, such systems are sometimes embedded in systems that certainly do function as a medium for communication between humans. In some plant control systems there is an intricate interplay between machine-generated and human-generated signals.

3.2. Conclusion

Semiotics and informatics share several common features. Our analysis has shown:

- that the application of edp may entail considerable changes in the language of the professional groups involved;
- that semiotics may contribute to new knowledge which may be utilised constructively in the development of edp-based systems.

Future work in this field could aim at developing new methods for descriptions of work processes and systems from a communication perspective.

FOOTNOTES

[1]) The following exposition is mainly based on Eco (1976), which again draws heavily on Hjelmslev (1961)

[2]) cf. Eco's "theory of sign production" and "theory of codes".

REFERENCES

Andersen, P.B., Nielsby, O. (1981). PROTEUS - en programmeringsfilosofi baseret pa selvforvaltning (PROTEUS - a Programming Philosophy based on Self-management). SAML 8. Institute of Applied and Mathematical Linguistics, Copenhagen.

Andersen, P.B., Kjær, A. (1982). Artificial Intelligence and Self-management. Journal of Pragmatics 3/4: 323-353.

Barthes, R. (1957). Mythologies. Seuil, Paris.

Bendix, E.H. (1966). Componential analysis of general vocabulary: the semantic structure of a set of verbs in English, Hindi, and Japanese. Indiana University, Bloomington.

Budde, R. et al. (1984). Approaches to Prototyping. Springer-Verlag, Berlin.
Bundgaard, J. et al. (1981). Kontor-Automations-Systemer (KAS) - Et studieprojekt (Office Automation Systems - a Study Project). Dansk Datamatik Center, Copenhagen.
Ciborra, C. (1981). Information Systems and Transaction Architecture. International Journal of Policy Analysis and Information Systems. 5.
Clark, H.H., Clark, E.V. (1977). Psychology and Language. Hartcourt Brace Jovanovich, New York.
The Danish Nurses Organisation. (1982). EDP- Handbook for Nurses. Public Services International. Feltham.
Eco, U. (1976). A Theory of Semiotics. Indiana University Press.
Floyd, C., Keil, R. (1983). Adapting Software Development for Systems Design with the User. Systems Design for, with, and by the Users. Briefs, Ciborra, Schneider (Eds). North-Holland, Amsterdam.
Hjelmslev, L. (1961). Prolongema to a Theory of Language. University of Wisconsin, Madison.
Hjelmslev, L. (1959). Essais linguistiqies. Nordisk Sprog- og Kulturforlag, Copenhagen.
Larsen, S.F. (1980). Egocentrisk tale, begrebsstruktur og semantisk udvikling (Egocentrical Speech, Concept Structure, and Semantic Development). Nordisk Psykologi 32: 55-73.
Larsen, S.F. (1981). Knowledge Updating. Psychological Reports. Institute of Psychology, University of Aarhus, Aarhus.
Larsen, S.F. (In press). Specific Background Knowledge and Knowledge Updating. Foregrounding Background. Allwood and Hjelmquist (eds). Doxa, Lund.
Lyons, J. (1977). Semantics. Cambridge University Press, Cambridge.
Mathiassen, L. (1981). Systemudvikling og systemudviklingsmetode (Systems Development and Systems Development Method). DAIMI PB-136. Department of Computer Science, University of Aarhus, Aarhus.
Nurminen, M.I. (no date). In Search for the Purpose of Information in Information Systems. Institute of Information Sciences, Bergen.
Sandewall et al. (1982). Stepwise Structuring, a Style of Life for Flexible Software. Software Systems Research Center, Linköping.
Searle, J.R. (1969). Speech Acts. Cambridge University Press, London.

WHAT DOES REAL WORK ANALYSIS TELL US ABOUT SYSTEM DESIGN?

Leonardo Pinsky and Bernard Pavard

Laboratoire de Physiologie du Travail
Conservatoire National des Arts et Métiers, Paris
France

Human-computer interface design is explicitly or implicitly based on specific representations of the user's activity. Several strategies are used for system design. Some of them are based on "user's models" (for example: Cuff, 1980), which are not usually precise enough to predict the numerous difficulties in the system use. Others rely on general principles to elaborate command languages or to structure the man computer dialogue. These general principles may be:

(a) the simplicity and coherence of the commands,
(b) the "natural" aspect of the actions during dialogue,
(c) the personalisation of the human-computer interface.

The design of command languages based on an analogy with natural languages, whether verbal (Treu, 1982; Landauer et al., 1980; Ledgard et al., 1980), graphic (Buxton, 1982) or musical (Buxton et al., 1983) does not systematically lead to solutions which will be appropriate for the functions the language is to fulfill (Fitter, 1979).

In general, these principles come from either informal observations of the use of different systems, or from the designer's intuition (Treu, 1976). This procedure may result in an incorrect representation of the operator's real activity and thus in models which have little to do with reality, leading to errors in the design. A correct representation of the activity is needed from the appropriate which characteristics of the system to be designed can be deduced. Several attempts have been made to develop general models for particular cases. Card et al., 1983, used a very detailed model of the activity of an operator while working on a word-processing task. Although this model provides precise data on the operator's behaviour, it only concerns tasks for which most of the aspects of the activity have been defined in advance. Although analysis of procedural errors is essential for the improvement of a system, this model cannot predict them.

This paper means to show that work analysis can help to elaborate activity representations wide enough to fulfill design requirements. Work analysis is not only concerned with performance, but also tries to define the structure of the activity as well as the cognitive processes of the operator within the real work situation. It aims at describing the complexity of the activity without making any a priori reduction. In order to gather relevant data from work analysis, the ergonomist must deal with a situation (man + computer + task) closely related to the one he has to design. In this case, the ergonomic work proceeds in degrees: the designer does several ergonomic experiments in the work place in order to improve the description of the functional characteristics of the system (see part one).

Work analysis can be integrated in a diffirent way: in the design process it can provide pertinent data by allowing experiments to take place in a laboratory, in order to study certain aspects of the cognitive processes involved in specific tasks. The difficulty associated with this approach depends on the choice of the variables used for the experiment. For validity's sake, the experiment must consider

constraints due to both the environment and the task. But, in fact, the environmental constraints which affect the cognitive processes are often difficult to define in advance. This experimental approach proceeds by "reducing" the real situation, but this "reduction" is made "a posteriori", that is to say: after identification of the constraints due to the environment. This paper describes two examples of the ergonomic research which is being undertaken in our laboratory, each within their own context:

(a) a close relation with a team of designers working on a particular on-line data coding system,
(b) a more general study aiming at the design of text composition systems.

Through these two examples, we will point out the two aspects of work analysis contribution to system design.

1. ON-LINE DATA CODING TASK

1.1. Circumstances of the ergonomic intervention

A system already existed to perform the on-line data coding (system I). The ergonomic team was assigned the task of helping the design of a new system (system II) which would be adapted to the operator. A first analysis of the real situation (system I) provided guidelines for the design of system II. Whilst designing this new system ergonomic experiments with volunteers were used to discover the features of the future work and its difficulties. The ergonomic experiment is defined in that way: the experimental situation is as similar to the future work situation as possible and the experimental protocol allows to control some variables (for instance, the same set of printed forms to be processed was given to the different operators).

1.2. Functioning principles of the on line data coding system II

The operators have to code the information gathered for a survey. We will consider the coding of the variable "profession": a list of professional categories was determined before the experiment began, each category having its code. Transferring data from printed forms to professional categories is a complex operation. The dialogue with the computer can be sketched as follows:
The operator transmits a screen-form:

PROFESSION	ELECTRONICAL TECHNICIAN	13-ST 4
14C-AE	CANCER HOSPITAL	
15A-CPF 5	15C-FONC 5	

This screen-form contains designations (electronical/technician) and coded values (13-ST 4).

The computer may reply in one of two ways:
- whenever a code is finally assigned, the computer sends back the name of the proper category.

```
PROFESSION      ELECTRONICAL TECHNICIAN      13-ST 4
14C-AE          CANCER HOSPITAL
15A-CPF 5       15C-FONC 5
EE27 : MAINTENANCE TECHNICIANS, ELECTRICITY, ELECTRONICS, AUTOMATISM REPAIR.
```

- otherwise, it sends back a message:

```
PROFESSION      MAINTENANCE TECHNICIAN      13-ST 4
14C-AE          CANCER HOSPITAL
15A-CPF 5       15C-FONC 5
MAINTENANCE TECHNICIAN, REPAIR: ASSIGN A VALUE FOLLOWING THE SPECIALITY
ELECTRICITY, ELECTRONICS, RADIO, T.V. ENTER C-EE27
MECANICS                              ENTER C-ME23
BUILDING, PUBLIC WORK                 ENTER T-BT1
```

The operator must choose one term from the message given.

To facilitate access to the list of designations, the designation used by the system does not necessarily consider all the words TYPED BY THE OPERATOR. For example, the operator transmits:

SUPERIOR TECHNICIAN IN ELECTRONICS AND AUTOMATISM

The system will use:

[ ] TECHNICIAN ELECTRONICS [ ]

Such a designation begins and ends with blanks at the first and the fourth places.

1.3. Work analysis

The operator using the system makes a decision by choosing a category. To do so the operator instigates actions, that is to say her behaviour is intentional, conscious, planned and goal directed (see Von Cranach, 1982). In fact, her behaviour is not only a reaction to the system answers. These actions and their sequence are guided by a reasoning which is not explicited by the data appearing on the display. Therefore, we added verbal data collected during the operator's work (verbal protocol).

1.4. Description of the operator's reasoning

In order to describe the operator's reasoning, we defined original notions by taking our inspiration from the "Natural logic" of J.B. Grize (1982). We hereafter give some characteristics of the operator's reasoning we were able to evidentiate:

(a) Objects

The operator considers two kinds of objects

- the profession of a given person, as it results from the totality of data collected from the printed forms,
- the objects sent back by the system: names of the categories and terms of the messages.

These objects are discursive entities worked out by the operator who will extract some information end/or combine the selected information with other information.

(b) Operations

The operator will use two types of operations on these objects:

- She will bring together different objects. For example:

Profession:

SHEET-IRON WORKER WORKING FOR THE POST-OFFICE

Message:

SHEET-IRON WORKER FOR CARS, MOTORCYCLES, TRUCKS	ENTER CME44
INDUSTRIAL SHEET-IRON WORKER	ENTER CME42
SHEET-IRON WORKER FOR VENTILATION, DECORATION, BUILDING	ENTER CME64

Verbal protocol:

"Knowing that he works for the post-office, I suppose he repairs the cars of the Company. It is certainly the case, so I will assign the following code: CME44".

The operator brings together one term of the message and the profession, on the basis of a common property: "cars repair".

- She will differentiate between objects, for instance:

Profession:

PASTRY-COOK APPRENTICE

Name of the category:

PASTRY-COOKS OR BAKERS (EXCEPT INDUSTRIAL ACTIVITY)

Verbal protocol:

"It does not fit, because the person is only an apprentice".

The operator differentiates between the name of the category and the profession because of the absence of the term "apprentice".

These two operations constitute a first step towards the elaboration of an action. For example, when "bringing together" different objects, the operator will tend to accept a category or take into account the term of the message. However, other operations can be made by the operator upon "bringing together" or "differentiating" operations. In particular, a differentiation can be reduced, neutralised or just left aside during operations dealing with something other than objects, such as the system functioning or the structure of the list of categories. At this moment interpretation of the system answers takes place (this interpretation has been described in a previous paper (Pinsky, 1983)).

1.5. Criteria for ergonomic improvements design

The analysis of the operator's reasoning shows the difficulties caused by an inappropriate system. We will distinguish three types of difficulties:

(a) Weakness of the reasoning due to an inadequate answer from the system (in general, a lack of information):
For instance, certain actions of "bringing together" are "poor": the operator does it on the basis of a unique word present in the profession to be coded and in the answer of the computer, as she has no other data at her disposal.

(b) Lack of structure of the information to be memorised:
More often than not, the operator cannot use the results of previous "differentiating" or "bringing together" operations for generalisation. Consequently, she is obliged to work, not with structured subject matter, but with a certain number of specific solutions, case after case, which are very difficult to memorise. This situation may even prevent the operator from learning how to use the system properly.

(c) Parasite activities:
The system is supposed to help the operator solve the coding problem. But, in certain cases, the operator is obliged to perform a series of actions to cope with the inadequacy of the system answers, without having explored the possibilities embedded in the list of categories. Such actions will be referred to as "parasite activities".

The modifications which will be needed to improve the ergonomic value of this system will be devised not only to eliminate these types of difficulties but also to provide some form of assistance to operators confronted with codification problems.

1.6. Some examples of ergonomic improvements

(a) Marking the words of the designation actually used by the system:
This supplies the operator with additional information and prevents those parasite activities which were caused by having to identify the words used by the system. It also widens the scope of possibilities of the two "operations" bringing together and differentiating, thus helping the decision process.

(b) Contents of the names of the categories and of the terms of the messages:
The analysis of the different methods of bringing together or differentiating provides general principles for the formulation of these items.

(c) Command of differentiation between categories:
The operator must often choose between two categories (she has performed two bringing together operations). It is not always possible, when editing the terms of the messages or the names of the categories, to really differentiate between two categories. We have therefore proposed a two argument-command (with the codes of the two categories) which produces a message expressing the differences between the two.

2. DESIGN OF A WORD PROCESSING SYSTEM

2.1. Word processing

Word processing is a general term for several cognitive activities which are very different in their nature: typing, correction, re-writing, translation, typographic editing, etc... These cognitive activities, as we will see later, are not only determined by the goals of the task to be done, but also by the characteristics of the technical system used by the operator. The interaction between technical systems and psychological processes was studied during a letter composition task in which the subject of the letter was supplied by the experimenter (Gould, 1978; 1982). Gould sees the composition activity as a succession of planning and production periods. During the planning periods, the author organises his discourse mentally, whereas during the production periods, he generates the same discourse written or spoken. Gould observed that the planning period is shorter (in time) when the production is spoken rather than written. The organisation of the sequences of planning and production periods depends on the choice of the author: a written discourse implies a very precise sequence of planning and production periods, a spoken one allows these two kinds of activity to merge. Although the organisation of composition activities (planning, production) varies with the choice of a written or spoken edition, the semantic or stylistic contents of the final product do not seem to be influenced by this choice, (Goldberg, 1979, cited in Gould, 1982). But, on the other

hand, an analysis of a redaction task carried out by journalists of a press agency (Duraffourg et al., 1982) showed that the use of type-writers or visual displays did influence the cognitive processes during dispatch redaction. The introduction of visual displays resulted in an increase of dispatch production and a decrease in quality. First the analysis of text composition procedures will be used to demonstrate how the introduction of visual displays lead the journalists to re-structure their sentence planning activity. Then the relation between the editing constraints of a technical display and the structure of the final text will be examined on the basis of the results of an experiment.

2.2. Analysis of redaction processes

In order to gain insight into the redaction processes, every sequence of words and/or characters processed on the display was recorded. The analysis of such a linguistic production allowed us not only to compare the initial text with the final product, but also to track down the intermediary attempts which did not result in a solution which was acceptable to the journalist. To simplify the analysis, we split the sentences up into propositions, and the propositions into components. A proposition, by definition, is composed of a theme (subject), a predicate (verb) and one or several arguments. We made a distinction between main arguments whose presence is necessary and secondary arguments which are not essential. The secondary arguments are often used to specify the temporal or spatial conditions of the main argument. For example: "John buys some bread at the baker's". We successively find: the theme (John), the predicate (Buys), the main argument (some bread) and the secondary argument which indicates the place where the action is performed (at the baker's). An example will be used to illustrate the different strategies used by the operator to plan and write a sentence. These strategies cannot be detected unless the journalist uses a visual display. The journalist's task will in this case consist of merging the two following sentences into a unique sentence: "The principal leader of MIR and his seven collaborators have been killed yesterday during a fight with the police. According to the official version, the fight took place when the government security service got informed of a meeting held by the MIR's staff". Figure 1 shows the first attempts of the journalist to merge these two sentences.

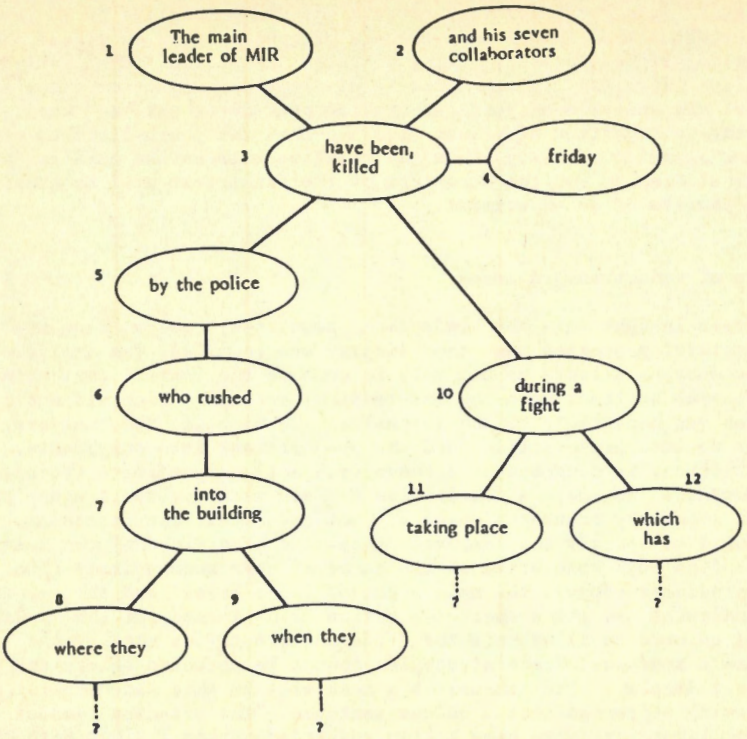


Figure 1 First redactional attempts of the journalist. The numbers indicate word-processing sequence

The numbers indicate the sequence of redaction. We can see that the journalist undertakes the redaction process without planning more than two propositions. Figure 2 shows a new attempt of the journalist to compose his sentence, and the final solution. The components outside the circled areas have been inserted between the components already written on the visual display.

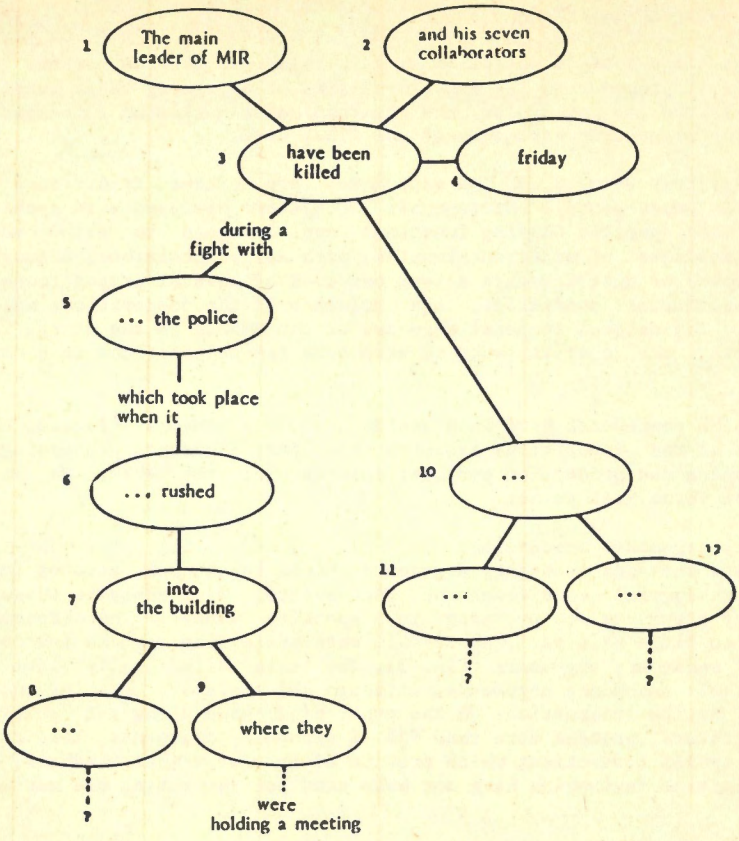


Figure 2 In order to re-write his text, the journalist "inserts" (words outside circled areas) or suppresses some elements of a given linguistic material

This simple example shows that a text processing display induces specific redactional procedures:

(a) the planning of the composition is a short-term activity (one or two propositions only);
(b) if a sentence is incorrect, the journalist avoids re-writing it entirely. He prefers to adopt a problem-solving strategy and to choose the lexical elements likely to be inserted in the written text available.

Other strategies of composition have been observed. The journalist, for instance, first plans his sentence without the secondary arguments and then, when re-reading it, adds these by using the "insert" function.

2.3. Experimental approach

Work analysis allowed us to establish certain relations between system constraints and redactional procedures. The experiment aims at analysing these constraints more systematically and, in particular, the relation between editing functions, the ease of use and the linguistic structure of the final text.

The editing systems we used for the experiment were either traditional ones like type-writers, paper-pencil, dictaphone, or systems equipped with specific editing functions. These specific editing functions are designed to allow working with linguistic entities of different lengths, such as: propositions, arguments, words, etc... The operator must re-write a text composed of several propositions. Depending on the experimental conditions, the sequence of the propositions may or may not correspond to the natural temporal sequence of the events in the story. To compose a coherent text, the operator must re-structure the propositions in a chronological sequence.

The composition procedures have been analysed at four levels (Pavard, 1984): re-structuring of the propositions sequence; secondary arguments processing; organisation of planning and production periods; coherence of the text. We will shortly analyse these three last points.

(a) Secondary arguments processing:
A sentence contains secondary arguments which, in general, account for the temporal or spatial conditions of the events. Work analysis shows that these secondary arguments are processed in a specific manner. The experimental approach confirms this fact, as certain word-processing systems induce the "omission" of secondary arguments (Fig. 3). The texts edited orally lack more than 1/4 of their secondary arguments, although the operators were instructed to give back the entire information. On the other hand, when using the "puzzle" editor, the operators process more than 95% of secondary arguments. This effect is due to this system's functions which provide additional visual feedback: those parts of the initial text which have not been used for re-writing are marked.

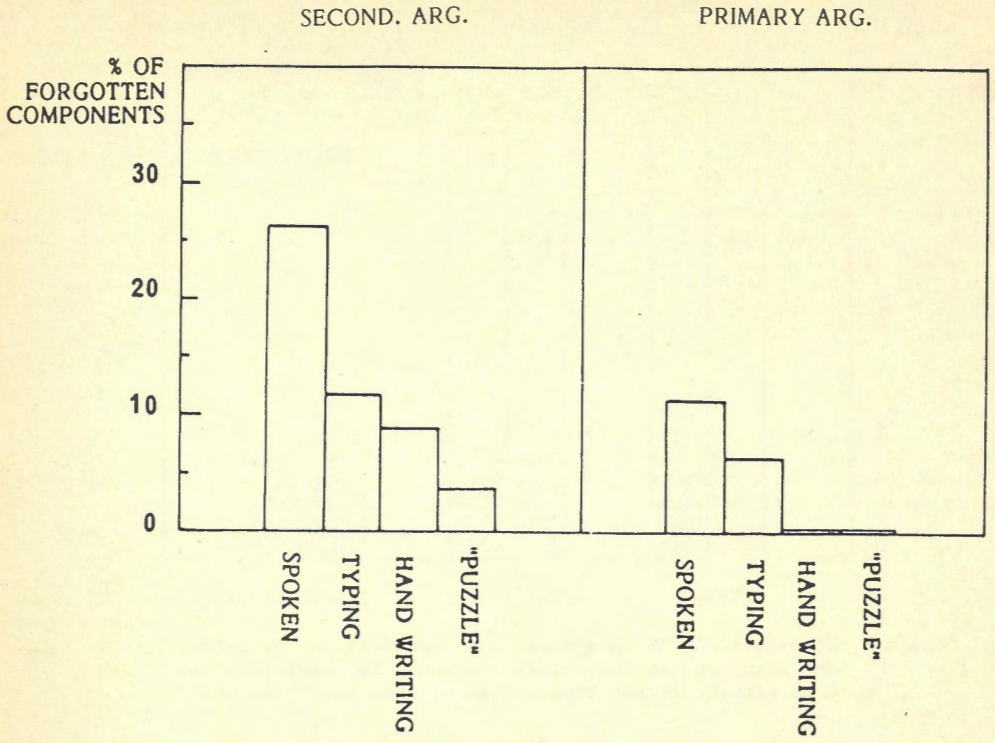


Figure 3 Percentage of sentence components "omitted" during rewriting activity, depending on the text edition system

(b) Organisation of the planning and production periods:
The amount and kind of organisation depends on the text editing system employed. Our results are in accordance with those of other studies. Spoken composition is the quickest (Fig. 4). The orator plans his text both before and during the production period. Although the coherence of texts orally processed is similar to the one of texts processed with other methods, the operator only considers a subset of the secondary arguments involved. This particularity of the spoken edition may be due to the absence of a physical representation of the text produced. The operator can only process a limited number of propositions or arguments at a time.

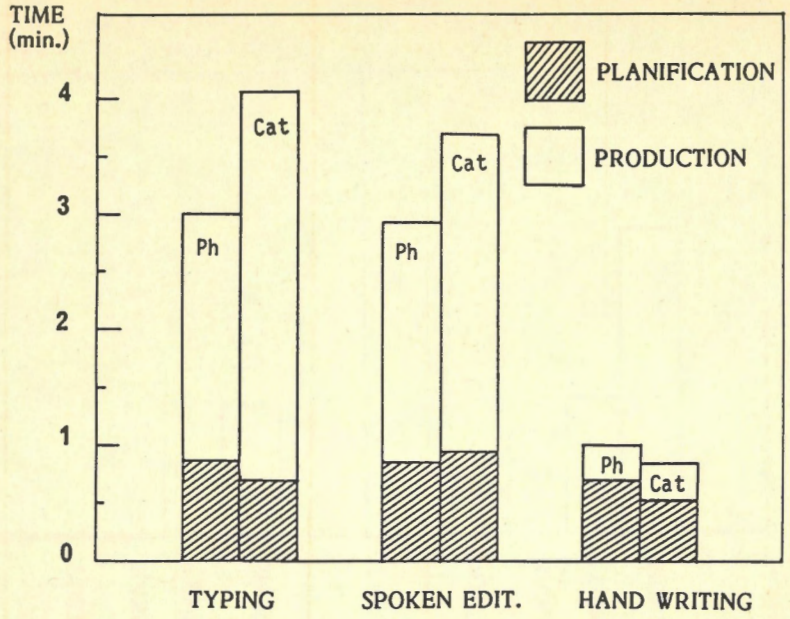


Figure 4 Organisation of planning and production periods, depending on the text edition system. Two texts (Ph and Cat) with different thematic structures have been used.

(c) Coherence of the text:
Among the quality criteria for a text, inter-propositional coherence is one of the most important. In order to be acceptable, a text must be composed of propositions linked together, either explicitly or implicitly. Nevertheless, a text whose propositions are linked together by "connectors" is not necessarily correct in terms of semantics. There are specific rules which limit the possible combinations of several "connectors" (a connector is a grammatical element which connects different propositions, such as: and, because, in order to, as, etc...). The operator must respect these rules, or else he will produce a text which is semantically unacceptable. For instance, we could observe a competition effect between the respect of text coherence and the respect of connector combination rules. Some systems induce composition strategies within which coherence is the main concern, at the expense of semantical acceptability.

3. CONCLUSION

These two studies have shown how work analysis focuses on the complexity of activity. Text composition is not only analysed in terms of the use of editing functions, but more widely, in terms of psycho-linguistic processes. On line data coding is seen as a process of elaborating a decision in accordance with the system's answers, and not only as a sequence of transactions. The work analysis methods we used show two ways to find the relation between cognitive processes and characteristics of the system. They both evolve from the recording of the operator's behaviour. In one case, we studied the unconscious psycholinguistic procedures, through a comparative method (several text composition systems were used).

In the other case, we added verbal information to the behaviour recordings, thus focusing on the conscious reasonings underlying the actions and defining their relation with the use of a display. These two studies also show the importance of experimentation for system design when based on work analysis. It can be most useful, either during the design process, by using a series of ergonomic experiments in situations which resemble the future work situation as much as possible, or in the laboratory, provided that its results will be replaced in the context of the real work situation, which can be done thanks to a previous work analysis.

REFERENCES

Buxton, W., Sniderman, R., Reeves, W., Patel, S.,Baecker, J. (1979). The evolution of the SSSP score editing tools. Computer Music Journal, 3, 4, 14-26.

Buxton, W., (1982). An informal study of selection positioning tasks. Graphics Interface 82, 323-328.

Buxton, W.,Lamb, M.R., Sherman, D., Smith, K.C. (1983). Towards a comprehensive user interface management system. Computer Graphics, 17.

Cranach Von, M. (1982). The psychological study of goal-directed action: basic issues. The analysis of action - recent theoretical and empirical advance. Von Cranach M., and Harre R. (eds). Cambridge University Press.

Cuff R.N. (1980). On casual users. International Journal of Man-Machine Studies, 12, 163-187.

Duraffourg, J., Guerin, F., Pavard, B., Dejean, P.H., Launay, F., Pretto, A., Vladis, A. (1982). Informatisation et transformation du travial. A.N.A.C.T., Paris.

Fitter, M. (1979). Towards more "natural" interactive systems. International Journal of Man-Machine Studies, 11, 339.

Gould, J.D. (1978). An experimental study of writing, dictating, and speaking. Attention and performance VII. Requin J. (Ed.). Erlbaum and Assoc., Hillsdale. New Jersey. 299-319.

Gould, J.D. (1982). Writing and speaking letters and messages. International Journal of Man-Machine Studies, 16, 147-171.

Grize, J.B. (1982). De la logique à l'argumentation. Droz, Genève, Paris.

Landauer, T.K., Galotti, K.N., Hartwell, S. (1980). A computer command by any other name: a study of text editing terms. Bell Laboratories Report.

Ledgard, H.F., Whiteside, J.A., Singer, A., Seymour, W. (1980). The natural language on interactive systems. Communications of the A.C.M., 23, 556.

Pavard, B. (1984). Approche ergonomique de la conception de systèmes de traitement de textes, in Rapport C.N.R.S. - ATP-955 III, Paris.

Pinsky, L. (1983). What kind of "Dialogue" is it when working with a computer? The psychology of computer use. T.R.G. Green, S.J. Payne, G.C. van der Veer (eds). Academic Press, 29-40.

Treu, S. (1982). Uniformity in user-computer interaction languages: a compromise solution. International Journal of Man-Machine Studies, 16, 183-210.

PSYCHOLOGICAL SELECTION OF PERSONNEL FOR DATA PROCESSING PROFESSIONS

Horia Pitariu

Universitatea Cluj-Napoca
Roumania

The introduction and implementation of computers in Roumanian society have been phased, following a programme which will result in a national date processing system. In order to achieve this goal electrical data processing equipment is being constructed and training programmes are being given to "selected" candidates. These training programmes include intensive courses for analyst-programmers, junior programmers, card-punch and computer operators. The psychological selection method by which the trainers are chosen from the large number of candidates with their diverse backgrounds, is the subject of this study.

1. PSYCHOLOGICAL SELECTION OF ANALYST-PROGRAMMERS AND JUNIOR-PROGRAMMERS

Psychological studies of analysis and programming activity have been made by authors from various countries (Hoc, 1974-1975; McNamara and Hughes, 1961; Moulin, 1971; Lindell, 1978; Strizenec, 1973; Szafraniec, 1975). Briefly, the analyst is a person who formulates the problem - sometimes very spontaneously and informally -, interprets the information presented as data, gives it a form and assigns it a processing method. He has a good knowledge of data processing, a broad basic background and may be specialised in a particular field of application (Arsac, 1970). Programming is restricted to the activity of writing a programme as economically as possible. The novelty of approaches and solutions is characteristic of analyst-programmer's function, as opposed to standardised work schemes. Investigation has shown that certain psychological factors are common to the personalities of both the analysts and the programmers, whilst the analyst's personality has shown to be more varied and complex.

1.1. Subjects sample

The research was carried out on a sample of 224 subjects, participants in post-graduate data processing courses: 28% economists, 36% engineers and 37% teachers (physicists and mathematicians). The engineers were from different industrial units each with their own speciality and knowledge of the problems specific to their unit which is very important for the analyst/analysis job. The age of the participants was 28.7 years $\pm$4.8. Seniority ranged from 1 to 15 years.

The set of participants in the courses for junior programmer jobs consisted of 69 subjects, their average age being 28 years $\pm$6.7.

1.2. Job performance ratings

Without sticking to one type of criterion, it is to be emphasised that our orientation was to detect a proximal, "static" criterion, with a general, multidimensional character, but with possibilities for being changed into unidimensional one,

according to the concrete conditions offered by the research. To achieve this, the grades obtained by the students during the schooling period in knowledge tests, project and final examination, as well as a rating scale were used.

1.3. Choice and validity of psychological tests

The test battery we chose is the Computer Programmer Aptitude Battery (CPAB), worked out by J.M. Palormo (1967). The battery consists of five subtests: Verbal Meaning (VM), (a test of communications skill; vocabulary commonly used in mathematical, business, and system engineering literature), Reasoning - R (an ability test on translation of ideas and operations from word problems into mathematical notations), Letter Series - LS (a test of abstract reasoning ability, finding a pattern in the given series of letters), Number Ability - NA (measuring facility in using numbers; ability to quickly estimate reasonable answers to computations), Diagramming - D (measuring the ability to analyse a problem and order the solution steps in a logical sequence). The reliability coefficient of the total battery score is .95 (Palormo, 1967). The Domino 48/70 (D 48/70)(1961) was also added, and the Domino 48/70 and Abstract Reasoning (AR) (subtest of the DAT battery), tests for junior-programmers. Domino 48/70 is a nonverbal general intelligence test similar to Raven's Progressive Matrices. It is made up of 44 problems that consist each of a group of dominoes arranged according to a certain rule. The subject is required to find the rule for completing the arrangement of the series. The reliability coefficients are .89 (D 48) and .90 (D 70). Factor analysis of the test battery revealed the existence of two factors which covered 55% and 15% respectively, of the variance. The former corresponds to a general factor, analogous to Spearman's "g" factor, named "Intellectual potential" or "Efficiency in analysis programming"; the latter is a bipolar factor, "Verbal reasoning - serial-algorithmic reasoning".

The set of subjects which formed the object of our validation studies consisted of participants in the post-graduate and high-school- graduate course in data processing. The psychometric characteristics are mentioned in table 1.

	Analyst-programmers								Junior-programmers	
	Group I (n=40)		Group II (n=47)		Group III (n=64)		Group IV (n=57)		(n=69)	
Variables	M	SD	M	SD	M	SD	M	SD	M	SD
Age	25.50	3.32	28.29	4.24	28.84	5.65	28.30	5.43	28.00	6.71
CPAB - VM	18.89	7.27	14.47	5.99	14.83	5.26	14.70	6.14	10.07	4.74
CPAB - R	10.00	3.39	11.68	5.00	12.29	4.38	12.00	4.48	6.45	3.48
CPAB - LS	7.35	3.68	9.06	4.97	10.23	3.86	9.53	4.28	6.80	3.45
CPAB - NA	12.07	4.08	13.47	4.75	12.70	4.34	13.77	4.82	8.87	3.35
CPAB - D	15.95	6.67	16.94	9.08	16.66	7.57	16.86	8.66	8.65	5.84
Domino 48/70	27.93	6.01	28.55	6.91	28.06	5.23	27.25	5.12	24.28	6.26
DAT-Abstr.R.									34.25	8.27
Criteria	53.73	20.56	48.77	17.30	3.03	1.10	6.65	1.53	3.01	1.08

Table 1 Psychometric and criteria performance of the analyst-programmer and junior-programmer groups (In Group I & II, the criterion consisted of the overall score of the appraisal card, in Group III of forced-choice rating, and in Group IV, of combining the grades obtained in the course modules).

	Analyst-programmers								Junior-programmers	
	Group I (n=40)		Group II (n=47)		Group III (n=64)		Group IV (n=57)		(n=69)	
Test	r	R	r	R	r	R	r	R	r	R
CPAB - VM	.08	.08	.45**	.47**	.08	.09	.26*	.26*	.48**	.47**
CPAB - R	.46**	.57**	.49**	.49**	.41**	.42**	.30*	.30*	.55**	.39**
CPAB - LS	.46**	.55**	.22	.22	.30**	.35**	.37**	.39**	.28*	.29*
CPAB - NA	.19	.21	.35*	.35*	.39**	.41**	.19	.19	.29*	.29*
CPAB - D	.59**	.69**	.53**	.53**	.35**	.39**	.57**	.57**	.35**	.35**
Domino 48/70	.51**	.59**	.34*	.36*	.34**	.39**	.26*	.28*	.49**	.56**
DAT-Abstr.R.									.51**	.55**
Mult. corr.	.69		.61		.54		.58		.78	

Table 2 Correlations (r) of test performances with success in data processing courses and the correlations corrected for restriction of range (R). (* p=.05; ** p=.01)

In the validation of the predictors used for psychological selection, an instrument was derived which did not easily alter its characteristics and from which the predictions were as stable as possible despite the lapse of time. To achieve this, the validation of the test battery used for the selection of analyst-programmers included four successive cross-validations, corresponding to a similar number of post-graduate courses that were investigated. Table 2 is a synthesis of the prognostic value of the psychological tests. The analysis of the results yielded by the four cross-validations enables us to draw the conclusion that the VM and NA subtests are less reliable when compared to the rest of the tests. The overall validity of the battery, estimated by the multiple correlation coefficient, ranges from .54 to .69. For junior-programmers, the overall validity of the test battery was .78.

The gain achieved by using the psychological tests is 20% for analyst-programmers and 30% for junior-programmers. It is advisable that only those who have 70% (stanine 7) and respectively 79% chance of success (stanine 5) should be admitted to courses (Pitariu, 1977; Pitariu, 1978).

2. PSYCHOLOGICAL SELECTION AND TRAINING OF CARD-PUNCH OPERATORS

There have been some attempts at validating tests of psychological selection but in our situation they were not very successful and, due to the use of various types of card-punch machines, treated a diverse relationship between man and machine. The research in question is therefore concurrent with the setting up of an adequate selection and training system, beginning with a work place supplied with up-to-date equipment and tasks.

2.1. Psychological analysis of the profession

Initially psychological analysis of the work was to ascertain the specific errors made in the punching activity. Lahy and Korngold-Pacaud (1936) distinguish three categories of errors: gap errors - considered as most common, punch errors, reading errors, i.e. transposition of figures (figures wrongly remembered; incorrect reading owing to the document). Durey (1960) has emphasised five error types: reading, reversal, double punching, omission and shifting of fingers in different directions. A sample where alphanumerical material was used, enabled Kirchner and Banas (1961) to discover four main error types: spatial - 47% (pressing an adjacent key); perceptual - 25% (punching an "R" instead a "P"); reversal - 3% (punching a "25" instead a

"52" - the digits being reversed); sequential - 25% (the operator became one column behind or one ahead with the result that a whole sequence of data was punched into the wrong column). In another study, Kirchner (1966) regrouped errors in two types: perceptual - 56%; and spatial - 44% (r = .72). A work sample which we administered to professional operators revealed the following error types: perceptual errors (wrong reading of the source document) - 53%; sequential and reversal errors - 14%; and errors of operation (pressing of an adjacent key, wrong operation of "alpha/numeric" key) - 33% (Pitariu, 1976).

The analysis of punching errors points to their location in the perceptive motor-speed and attention fields. Further details can be obtained only by an analytic study of the operations carried out in the process of work.

A synthesis of the psychological work analysis data concluded that the persons suitable for a key-punch profession should be young high-school graduates with or without a diploma, between the age of 18 and 25, endowed with an average intelligence, capable of resisting monotony as well as concentrating attention, and possessing a good emotional stability. Persons having a good capacity for motor learning, hand coordination and finger dexterity will be preferred (persons with synkinesis cannot practise this profession, nor those with anatomopathologic distortions of hands or spline). Persons who are conscientious when accomplishing work tasks, who are receptive to oral or written instructions, and who easily accept supervision are also to be preferred.

2.2. Validation of test battery for key-punch operators

Ten tests were selected at first, from among which five have been retained as significant: Test of unremitting attention (UA)(Ricossay, 1960), Tapping (T), Test of distributive attention (DA), Test of simple arithmetic abilities (AA) and Domino 48 (D 48) (1961). The factor analysis of tests revealed three factors:

(a) capacity of effort, attention concentration in the conditions of an intense activity (explained 38% of the variance);
(b) perceptual structuring, opposed to finger dexterity (explained 23% of tne variance);
(c) a psychomotor factor (explained 19% of the variance).

The concurrent validation was initiated with a group of 50 key-punch operators 20 to 25 years old. To ascertain the validity of the tests, we have considered that a relevant criterion, fairly complete and capable of discriminating the operators from one another, can be set up by combining professional success reflected by the punching speed index and accuracy index, with estimations given by immediate supervisors. Consequently, a rating scale ranging from 1 to 10 was worked out. The arithmetic mean of the marks was used as a criterion. Psychometrical performance and validity coefficients are illustrated in table 3.
The test battery used for selecting key-punch operators has a multiple correlation coefficient of .65. In practice, the subjects "admitted" are those whose overall score is more than stanine 5, their average success chance being 85%.

Test	Mean	SD	r	R corrected
Test of unremitting attention:				
- rhythm imposed Slow+Fast	87.50	10.17	.44	.79
- rhythm imposed Fast	64.08	12.38	.36	.66
Tapping:				
- first pair of fingers (T_1)	53.35	9.31	.27	.25
- second pair of fingers (T_2)	52.15	8.03	.23	.24
Test of distributive attention	60.81	14.24	.39	.39
Test of arithmetic abilities	271.52	69.77	.25	.27
Domino 48	20.27	6.33	.23	.24
Multiple correlation			.65	
Regression equation: $X=40(UA_{s+f}) + 7(UA_f) + 15(T_2) + 21(T_1) + 18(DA) + AA + 22(D48) - 4748$				

Table 3 Results of concurrent validity study (n = 50)

2.3. Professional training of key-punch operators

Professional training of key-punch operators follows the psychological selection. The operators selected attend a course consisting of three modules:

(a) machine operation technique;
(b) introduction to automatic data processing;
(c) development of key-punching skills.

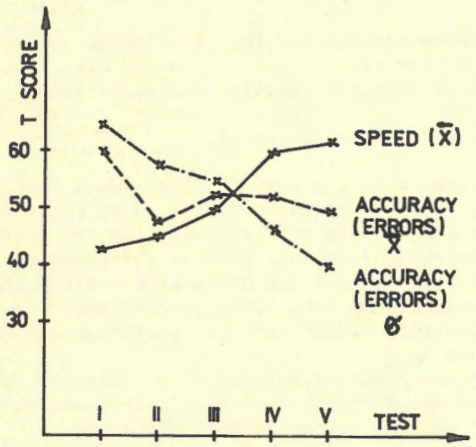


Figure 1 Performance of operators after 1 to 5 weeks

To develop key-punching skills operators practised key handling for a week and then continued their training with formative character exercises of the type recommended by Keys and Powell (1970) in addition to routine production activity. The exercises were based on repeating some patterns until automation is reached. This will afterwards facilitate the "blind" punching technique (e.g. Exercise 1: 4554 5455 4445 5545 4544 ...; Exercise 10: 4532 5679 9051 6750 ...) The exercises gradually became more and more difficult, progressively including more elements:

A A+B A+B+C A+B+C+D

Performance skills were evaluated by tests at the end of each week. Figure 1 illustrates the average performance of the operators. Between the first and the last performance evaluation test we obtained an efficiency increase of 43% ($t=4.21$; $p<.01$), achieving a final performance of 8080-11040 key strokes per hour. Errors have been reduced to 4% ($t=3.76$; $p<.01$) (Pitariu, 1980).

The strategy of selection and professional training of key-punch operators is illustrated by the flow chart in Figure 2.

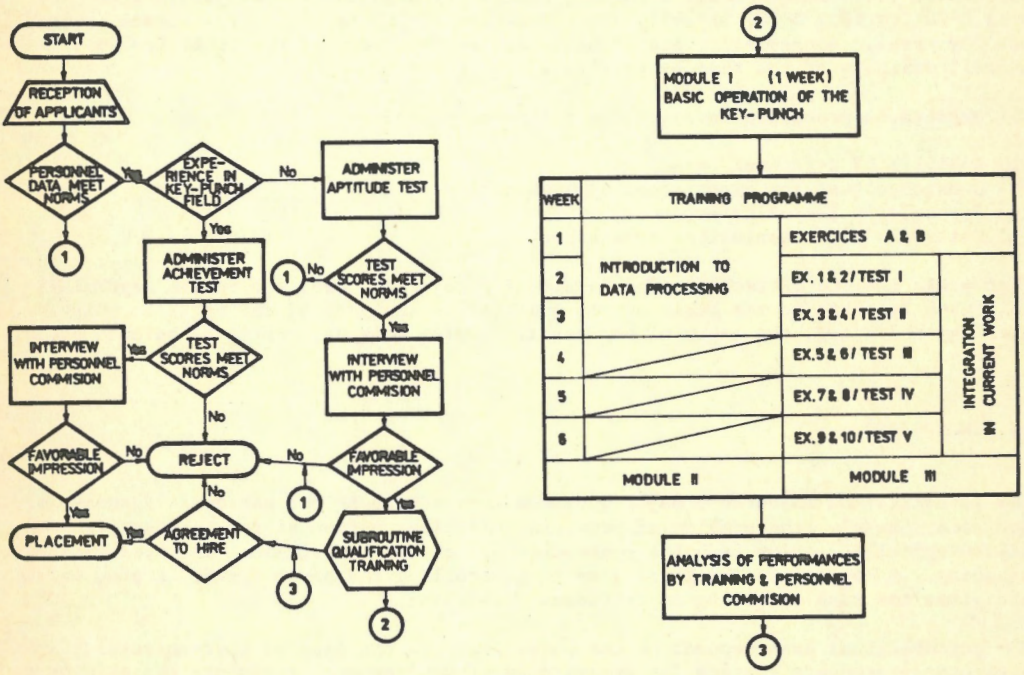


Figure 2 Flow chart of key-punch personnel selection and training process

3. SELECTION OF COMPUTER OPERATORS

A detailed analysis of the computers operator's activity showed that supervising the system operation consumed 26% of his time and work organisation 26%. The operator appears to perform a sequence of perceptive-motor actions. In fact these perceptive-motor actions are the consequence of a complex decision process resulting from the work to be performed and the capabilities of the computer. A few of the psycho-physiological traits and attributes that can be used to characterise an operator and which formed the basis of the psychological test battery used for selection purposes are: integrity of the visual and auditory senses, accuracy and decisiveness of eye-hand coordination and of motor response. The operator is characterised by a keen sense of observation and a short response time in decision making, in a tense atmosphere with frequent conflict situations associated to the system

functioning. From an intellectual point of view, the operator's psychological qualities are similar to those of the programmer's (clarity and flexibility of thinking, logical planning of actions, fine analytical-synthetical capacity etc.).

Validation of the psychological tests used in selection is a difficult task because of the restricted number of operators (N = 21, mean 25.6 years). We only used psychological tests that were administered individually to facilitate the clinical analysis of the data obtained. Thus, the psychometric score has played a secondary role, the behaviour analysis being used as the selection basis (thinking strategy, logical solutions to problems of a non-verbal character, emotional equilibrium etc.).

Of the tests selected, the following have been retained as having a significant validation (the criterion was a ranking of the performance of the operators on their job) B-20 (r=.50); B-101 (r=.45); SNB - duration of errors (r=-.47) (tests worked out by Prof. Bonnardel); and Finger dexterity (test of the GATB) (r=.46). The overall validity of the test battery is $R_{1(2345)} = .69$.

The employment procedure involves the following phases:

(a) Analysis of personnel data;
(b) Knowledge test for experienced applicants;
(c) Psychological examination;
(d) Interview with examination commission.

Each phase must be satisfied. The "admitted"/"rejected" decision in the psychological exam is taken on the basis of the qualitative analysis of the profile: thinking strategy and speed, the logic of non-verbal problem solving, emotional balance etc.

4. CONCLUSIONS

The investigation answered a major question that confronts the national economy of Roumania, namely the problem of providing efficient personnel for informatics. The psychological selection is but a subroutine of a vast programme of professional training. It represents the first step in controlling the human factor, a step which minimises the risk of making an erroneous prediction.

The psychological exam depends on the objective. In the case of post-university and high-school-graduate courses for analyst/junior-programmers' training, the objective of the selection exam was to eliminate the persons unsuitable for this job and to organise a homogeneous group capable of facing the rigour of an intensive course. Of course, when hiring trained personnel for analyst-programmer or junior-programmer jobs, the ability examination is of much lower importance than the psychological exam which is centered upon other functions and is correlated to professional use.

Concerning key-punch operators and computer operators the psychological examination plays a prominent role, the psychological selection being necessary because that the professional training is carried out on the job. When hiring "selected" personnel, a psychological exam is still useful, with the emphasis shifting in this context to the candidates' capacities for socio-professional integration in the work team.

REFERENCES

Arsac, J. (1970). La science informatique. Dunod, Paris.

Durey, D. (1960). Formation professionnelle fondée sur l'analyse de travail. La formation des mécanographes. Psychologie Française, V, 187-212.
Hoc, J.M. (1974). Quelques remarques sur l'analyste-programmeur. Bulletin de Psychologie, 318, 857-859.
Keys, W.J., Powell, C.H. (1970). A Handbook of Modern Keypunch Operation. Canfield Press, San Fransico.
Kirchner, W.K. (1966). Analysis and Prediction of Performance of Experienced Keypunch Operators. Journal of Industrial Psychology, IV, 48-52.
Kirchner, W.K., Banas P. (1961). Prediction of Keypunch operator performance. Personnel Administration, 24, 34-26.
Lahy, J.M., Korngold-Pacaud S. (1936). Sélection des opératrices des machines à performer "Samas" et "Hollerith". Le Travail Humain, IV, 280-290.
Lindell, J. (1973). Analysis of Computer Programming Work. Reports from the Department of Psychology, No. 545. The University of Stockholm.
McNamar, W.J., Hughes J.L. (1961). A Review of Research on the Selection of Computer Programmers. Personnel Psychology, 14, 39-51.
Moulin, M. (1971). La sélection des analystes-programmeurs dans l'administration des postes et télécommunications. Thèse, Paris.
Palormo, J.M. (1967). Computer Programmer Aptitude Battery. SRA, Chicago.
Pitariu, H. (1976). Psychological Selection of Keypunch Operators. Revue Roumaine des Sciences Sociales, Série de Psychologie, 20.
Pitariu, H. (1977). La sélection psychologique du personnel pour les professions du domaine de l'informatique. Le Travail Humain, 40, 131-140.
Pitariu, H. (1978). Psychological Selection of Analyst-Programmers and assistant-programmers. Revue Roumaine des Sciences Sociales, Sèrie de Psychologie, 22, 185-198.
Pitariu, H. (1980). Une expérience de sélection et de formation professionnelle rapide des opératrices mécanographes. Bulletin de Psychologie, 344, 431-436.
Ricossay, G. (1960). Etude sur le test d'attention soutenue. La machine-outil Française, 153-158.
Strizenec, M. (1971). Psychological Analysis of Work and Selection of Computer Operators and Programmers. Studia Psychologica, 15, 194-205.
Szafraniec, H. (1975). A Preliminary Job Analysis of the Computer Programmer and System Analyst. Polish Psychological Bulletin, 6, 217-226.
Test D 48. (1961). Centre de Psychologie Appliquee, Paris.

CONTRIBUTORS

Allard, R. Centre for the study of knowledge processes, Uppsala University, Box 227, S 75104 Uppsala, Sweden.

Andersen, P.B. Computer Science Dept., Aarhus University, Ny Munkegade, DK 8000 Aarhus C, Denmark.

Beishuizen, J.J. Subfaculteit Psychologie der Vrije Universiteit, De Boelelaan 1115 Prov. I. C113, 1081 HV Amsterdam, Netherlands.

du Boulay, B. Cognitive Studies Programme, Arts Building, University of Sussex, Falmer, Brighton, UK.

Cavallo, V. Laboratoire de Psychologie de l'Apprentissage, CNRS, IBHOP, rue des Gèraniums, 13014 Marseille, France.

Clegg, C.W. MRC/ESRC Social and Applied Psychology Unit, Department of Psychology, The University, Sheffield, S10 2TN, UK.

Deutsch, C. Société OPEFORM, 1 Rue de la Tour, 92240 Malakoff, France.

Falzon, P. Institut National de Recherche en Informatique et en Automatique, Domaine de Voluceau. B.P. 105, 78153 Le Chesnay, France.

Fisher, G. Universität Stuttgart Institut fur Informatik, Herdweg 51, D-7000 Stuttgart, Germany.

Gorny, P. Universität Oldenburg, Ammerländer Heerstrasse 67-99, D-2900 Oldenburg, Germany.

Green, T.R.G. MRC Applied Psychology Unit, 15 Chaucer Rd, Cambridge, UK.

Haring, G. Institut für Informationsverarbeitung, Technische Universität, Schiesstattgasse 4a, A-8010 Graz, Austria.

Hoc, J.M. Laboratoire de Psychologie du Travail de l'EPHE (ERA CNRS), 41 rue Gay-Lussac, 75005 Paris, France.

Kemp, N.J. MRC/ESRC Social and Applied Psychology Unit, Department of Psychology, The University, Sheffield, S10 2TN, UK.

Kommers, P.A.M. T.H. Twente - TO.-I.S.M., Department of Educational Research Postbus 217, 7500 AE Enschede, Netherlands.

Krasser, Th. Institut für Informationsverarbeitung, Technische Universität, Schiesstattgasse 4a, A-8010 Graz, Austria.

Lemke, A. Universität Stuttgart, Institut für Informatik, Herdweg 51, D-7000 Stuttgart, Germany.

Lind, M. Centre for the study of knowledge processes, Uppsala University, Box 227, S 75104 Uppsala, Sweden.

Mathiassen, L. Computer Science Dept., Aarhus University, Ny Munkegade, DK 8000 Aarhus C, Denmark.

Matthew, I. Department or Computing Science, University of Aberdeen, UK.

van Muylwijk, B. T.H. Twente, Afd. Toegepaste Onderwijskunde, Postbus 217, 7500 AE Enschede, Netherlands.

Oberquelle, H. Universität Hamburg, Fachbereich Informatik, Mensch-Maschine-Kommunikation, Schlüterstrasse 70, D-2000 Hamburg 13, Germany.

Pailhous, J. Laboratoire de Psychologie de l'Apprentissage, CNRS, IBHOP, rue des Geraniums, 13014 Marseille, France.

Pavard, B. Laboratoire de Physiologie du Travail, Conservatoire National des Arts et Mètiers, 41 rue Gay-Lussac, 71005 Paris, France.

Peruch, P. Laboratoire de Psychologie de l'Apprentissage, CNRS, IBHOP, rue des Gèraniums, 13014 Marseille, France.

Pinsky, L. Laboratoire de Physiologie du Travial, Conservatoire National des Arts et Mètiers, 41, rue Gay-Lussac, 75005 Paris, France.

Pitariu, H. Universitatea "Babes-Bolyai", Str. Kogàlniceanu 1, 3400 Cluj-Napoca, Roumania.
Potts, C. Department of Computing, Imperial College of Science and Technology, 180 Queen's Gate, London SW7 2BZ, UK.
Rohr, G. Software ergonomie, IBM-Science-Centre, Tiergartenstrasse 15, D-6900 Heidelberg, Germany.
Sandblad, B. Centre for the study of knowledge processes, Uppsala University, Box 227, S 75104 Uppsala, Sweden.
Schneider, W. Centre for the study of knowledge processes, Uppsala University, Box 227, S 75104 Uppsala, Sweden.
Schwab, T. Universität Stuttgart, Institut für Informatik, Herdweg 51, D-7000 Stuttgart, Germany.
Tauber, M.J. Software ergonomie, IBM-Science-Centre, Tiergartenstrasse 15, D-6900 Heidelberg, Germany.
Traunmüller, R. Johannes Kepler Universität Linz, Institut für Informatik, A-4040 Linz-Auhof, Austria.
van der Veer, G.C. Subfaculteit Psychologie der Vrije Universiteit, De Boelelaan 1115 Prov. I. B. 126, 1081 HV Amsterdam, Netherlands.
Waern, Y. Department of Psychology, University of Stockholm, Sweden.
Wall, T.D. MRC/ESRC Social and Applied Psychology Unit, Department of Psychology, The University, Sheffield, S10 2TN, UK.
van de Wolde, G.J.E. T.H. Twente, Afd. Toegepaste Onderwijskunde, Postbus 217, 7500 AE Enschede, Netherlands.
d Ydewalle, G. Department of Psychology, University of Leuven/Louvain, Tiensestraat 102, B -3000 Leuven, Belgium.