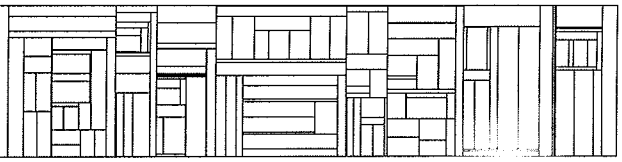
ISSN 0105-8517

Computer Support for Cooperative Design

Susanne Bødker
Pelle Ehn
Jørgen Lindskov Knudsen
Morten Kyng
Kim Halskov Madsen

DAIMI PB – 262
August 1988

AARHUS UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
Ny Munkegade 116 – DK 8000 Aarhus C – DENMARK
Telephone: + 45 6 12 71 88 *Telex: 64767 aausci dk*



COMPUTER SUPPORT FOR COOPERATIVE DESIGN*

Susanne Bødker
Jørgen Lindskov Knudsen
Morten Kyng
Computer Science Department,
Aarhus University, Ny Munkegade 116,
DK-8000 Aarhus C, Denmark.
Phone: +45 6 12 71 88.
E-mail: bodker@daimi.dk, jlk@daimi.dk,
mkyng@daimi.dk.

Pelle Ehn
Kim Halskov Madsen

Department of Information and Media Science,
Aarhus University, Niels Juelsgade 84,
DK-8200 Aarhus N, Denmark.
Phone: +45 6 13 67 11.
E-mail: ehn@daimi.dk, halskov@daimi.dk.

ABSTRACT

Computer support for design as cooperative work is the subject of our discussion in the context of our research program on Computer Support in Cooperative Design and Communication. We outline our theoretical perspective on design as cooperative work, and we exemplify our approach with reflections from a project on computer support for envisionment in design – the APLEX and its use. We see envisionment facilities as support for both experiments with and communication about the future use situation. As a background we sketch the historical roots of our program – the Scandinavian collective resource approach to design and use of computer artifacts, and make some critical reflections on the rationality of computer support for cooperative work.

INTRODUCTION

Design of computer applications and cooperative work will be discussed in two different ways. First we look at design as a process which may create the conditions for cooperation in use. Secondly, we look at the design process itself as one kind of cooperative work. To do so we identify and discuss the ideal that has become dominant in understanding cooperative work in and around the CSCW conferences: The small research group of the 1980s. Rooted in the Scandinavian tradition of designing in projects together with trade unions, we discuss some alternatives to the ruling ideal. We emphasize that it is important that designers of computer support for cooperative work do not just impose their own understanding or ideal of cooperative work onto other groups in

* Presented at the Second Conference on Computer-Supported Cooperative Work (CSCW'88), Portland, Oregon, September 1988.

other domains. Instead of heading for some ideal which may be more suited for the cooperation of researchers than for that of the users, we suggest that design is understood as a process which can help identifying and emphasizing future cooperation among the users.

Moreover, designers and users need tools and techniques to facilitate design as a cooperative process. We present our research program [4], in particular a part of it concerning computer support for cooperation among users and professional designers. Since design of computer support is design of the conditions for the future work situations of the users, these conditions need to be designed with concern for the practice and cooperation of the involved groups. We argue that an active participation of users in design is necessary to deal with this.

To be able to utilize the practical knowledge of the users and to be able to consider not only describable aspects of the computer support and future work situations, we advocate design by doing: A process of envisionment where the users can experience the future: Working with the (simulated) application. We present a computer-based object-oriented environment, APLEX, which is intended to support such cooperative design among users and professional designers. Finally we discuss some of the technical challenges deriving from the design of APLEX.

Hence, the paper start out with a broad introduction to our social and anthropological perspective on cooperative work, then focus on the design process as cooperative work, and finally zooms in on our more technical efforts to design computer support for this situation. In a concluding example we discuss the relations between these different levels of understanding computer support for cooperative work.

To set the stage, we will start out by making some critical reflections of the rationality of computer support for cooperative work.

RATIONALITY OF COOPERATIVE WORK

Briefly outlining the ruling ideal, we see a tendency to define the ideal for cooperative work as a small group of equally qualified people working together with very little managerial guidance or intervention. In other words, the ideal for a small research group in the 1980s.

One of the problems with the small research group ideal is that it is an ahistorical ideal. Conditions for scientific work have changed dramatically in this century. Researchers have been forced into large project teams where the outcome is partly determined on beforehand, and is to be achieved under great time pressure. The cooperative ideal seems to have developed in the same period of time as a way of preserving some of the freedom of the "real" scientist, who was before a creative loner in his study. The ideal of cooperative research work is something new, and it may change again, depending on the development of the conditions for research.

Few researchers have explicitly defined cooperative work, and those who do often base their definition on sharing of tools, materials and the like [14, 25, 26]. Is shared instruments a precondition for cooperative work, and does it make a difference if we talk about real-time sharing or sharing of a tool-box? Can office workers only work cooperatively if they also share typewriter, paper and pens? It is claimed that cooperation means no specialization or division of labor. As an example of the opposite, consider a woman giving birth to her child aided by a midwife, a nurse, and possibly a doctor. Specialization and division of labour is obvious, and at the same time it is definitely an example of cooperative work.

The definitions typically focus on cooperative work in general as an ideal, decontextualized from history, society and situation. The emphasis is on use of artifacts and materials, on communication and coordination of activities in general. We share this ideal with great sympathy. We think, however, that it is important to go beyond this abstract level, trying to understand computer support for cooperative work in a historical and social context – *to understand cooperative work in practice.*

By *practice* we refer to human everyday practical activity. In practice we produce the world. Both the world of objects and our knowledge about this world. Practice is both action and reflection. But practice is also a social and historical activity. As such it is being produced cooperatively with others, being-in-the-world. To share practice is also to share understanding of the world with others. However, this production of the world and our understanding of it takes place in an already existing world. It is the product of former practice. Hence, practice has to be understood socially: as our producing and reproducing social processes and structures as well as our being the product of them.

Rationality

The practical "reality" for cooperative work is often far from as rational and democratic as seems to be presumed in the "research group ideal". To get to a somewhat different conception, we will give two examples of totally different ways of looking at cooperation: *care rationality* – the "motherly" way of cooperation; and the *rationality of solidarity* – cooperation in the workers' collective. We do not introduce these examples to say that THEY are better ways of looking at cooperation than the research group ideal, but they definitely deal with communities where cooperation means something different.

Care rationality

In her book, *Caring. A Feminine Approach To Ethics & Moral Education* [20], Nel Noddings discusses caring as a philosophical alternative to the ruling ideal – "the father's voice" – the rational model, which is based on hierarchical thinking and abstract categories such as fairness and justice.

A person cares about somebody by taking on this person's situation, based on her former experiences with caring. Basically, we can only care about somebody because we, ourselves, have been exposed to the full-fledged experience

of being cared for at some point of time in our lives. The caring relation is, in other words, not a relation between equals, and we cannot just decide to care about each other as an explicit or implicit commitment.

The caring relation is a rather complicated one. The "one-caring" starts to care and determines how to act, not from an abstract categorization of different types of needs, but from making specific to herself the situation of the cared-for-to-be, and trying out that situation. In other words the rationality of caring relates to situations, whereas traditional, scientific, "male" rationality relates to abstract categories.

The main points which challenge our understanding of cooperative work are, first of all, that we deal with relations between people, which are mutual but not on equal terms. The degree of freedom to act, etc., is very different on the two sides. Secondly, commitment to participate in a caring relation from the one-caring's side is not sufficient. To be able to take on the role as the one-caring, the person must herself have been cared-for. Thirdly, the arationality and situation dependence of the caring relation, and the resolution of unresolved situations by prototypical investigations: the one-caring investigates prototypical situations based on the situation of the cared-for and tries out these prototypical situations – "how would I like it to be if my girl was in this situation...?"

Rationality of the workers´ collective

The sociologist Sverre Lysgaard has analyzed how factory workers together form a *workers' collective*; a support and protection mechanism against the ever ongoing exploitation by the managerial technical/economical system [18]. To Lysgaard, the workers' collective is a result of the tension between the individual on the one hand and the technical and economical exploitation on the other hand.

In a factory we would normally not call work cooperative. What Lysgaard describes is, however, a strong informal system which enable the workers to act together, instead of as out-standing and vulnerable individuals. The norms and values of the workers' collective become a buffer between the individual worker on the one hand, and the technical/economical system on the other. The conditions for this collective are not what the small research group ideal says: a multi-person tasks aided by technology, work done in an informal, normally flat organization, relatively autonomous. Rather it is determined by a complicated, dialectic relation with an inexorable, formal, hierarchical organization.

Different rationalities and cooperative work

Where the discussions about cooperation based on the small research group ideal have adapted "the father's voice" means-end rationality, the workers' collective represents a case of a different kind of rationality – the rationality of solidarity. This is still bound to contracts or commitments whereas the care rationality is an example of a kind of rationality which is not as directly bound

to commitment – the commitments can only be seen over time, as one person carries on the caring-for to another person.

Our point is that we do not need to see research work as an ideal for cooperative work to conduct a discussion of computer support for cooperative work. Rather we need to realize that there are many other ways of conducting cooperative work, and that these ways exists under a wide range of conditions; political, economical, gender-based, etc.

Cooperative work is many-folded and domain dependent. But still we believe in cooperative work, and we want to support situations of cooperative work. To build or introduce computer support for cooperative work is a process of change; not only of technology but also of the work place as such. Much in line with the Scandinavian collective resource approach we suggest that in trying to understand computer support for cooperative work we should supplement the focus on the "product" with a concern for the *design process*, including also the specific work practice and setting.

THE COLLECTIVE RESOURCE APPROACH

The *collective resource approach* in Scandinavia constitutes a major part of our background. It is based on two design ideals: The first is *industrial democracy*, the attempt to extend political democracy by also democratizing the work place – the social life of production inside the factory gates and office walls. The second is *quality of work and product*, the attempt to design skill-enhancing tools and environments for the production of highly useful quality products and services.

Both design ideals are of importance in the context of cooperative design. To have workers and designers cooperatively design skill-enhancing environments for users is a very direct way of having the workers influence their own work situation. Hereby, cooperative design contribute to industrial democracy.

Cooperation with the Trade Unions: The Scandinavian Projects

Practice along the lines of the collective resource approach has developed in Scandinavia during more than 15 years [9, 10]. In research as well as in design, the approach includes the workers who ultimately will be exposed to its results. The process was initiated in 1970 by the Norwegian Iron and Metal Workers Union (NJMF), which in cooperation with researchers from the Norwegian Computing Center embarked on a research project on planning, control and computerization from a trade union perspective. It was decided, as part of the project, to try out the work practices that the people in the project believed would become commonplace in the future: that the local unions themselves investigate their important problems at the work place and in the relation between the work place and the local community, and that in this work they use external consultants as well as internal consultants and other resources provided by the company.

The NJMF project inspired several new research projects throughout Scandinavia. In Sweden the DEMOS project on trade unions, industrial democracy and computers started in 1975 [8]. A parallel project in Denmark was the DUE project on democracy, education and computer-based systems [17].

Although growing, the extent and impact of these activities did not meet the initial expectations. It seemed that one could only influence the introduction of the technology, the training, and the organization of work to a certain degree. From a union perspective, important aspects like the opportunity to further develop skill and increase influence on work organization were limited. Social constraints, especially concerning power and resources, had been underestimated, and in addition the existing technology constituted significant limits to the feasibility of finding alternative local solutions which were desirable from a trade union perspective.

As an attempt to broaden the scope of the available technology, we decided to try to supplement the existing elements of the collective resource approach with union based efforts to *design* new technology. The main idea of the first projects, to support democratization of the design process, was complemented by the idea of *designing tools and environments for skilled work as well as for quality products and services*. To try out the ideas in practice, the UTOPIA project was started in 1981 in cooperation between the Nordic Graphic Workers' Union and researchers in Sweden and Denmark with experiences from the 'first generation' of collective resource projects [2].

The position we took in NJMF, DEMOS, DUE, UTOPIA and other collective resource projects was that decentralization of decision-making and a participative approach to the design process are not sufficient. Instead our position goes back to the different interests of management and workers concerning industrial democracy.

Conflicts and emancipation

Hence, we rejected the *harmony view* of organizations, according to which conflicts in an organization are regarded as pseudo-conflicts to be dissolved by good analysis and increased communication. Consequently we also rejected an understanding of design as fundamentally a rational decision-making process based on common goals. Instead our research was based on a *conflict view* of industrial organizations in our society. Within a conflict view it does make a difference whether you design cooperatively with management or with workers. In the interest of emancipation, we deliberately made the choice of working together with workers and their organizations, supporting the development of their resources for a change towards democracy at work. We found it necessary to identify ourselves with the "we-feeling" of the workers' collective, rather than with the overall "we-feeling" of modern management which focuses on gaining more productivity out of the work force. In short: Trade unions were seen as organizations with a structure that was problematic when functioning as vehicles for designing for democracy at work, at the same time

they were seen as the only social force that in practice could be a carrier of this ideal.

Human Centered Design

The political reason for involving end-users in the design process, and for emphasizing their qualifications and participation as resources for democratic control and change is only one side of the coin. The other is the role of skill and participation in design as a creative and communicative process.

This complementary concern has grown out of our dissatisfaction with traditional theories and methods for systems design – not only with how systems design has been politically applied to deskill workers, but more fundamentally with the theoretical reduction of skills to what can be formally described. Hence, one can say that the *critique of the political rationality* of the design process points to a *critique of the scientific rationality* of methods for systems description.

Our approach to cooperative design include users in a double sense. We claim the importance of rethinking the design process to include structures through which ordinary people at their work place more democratically can promote their own interests. We also claim the importance of rethinking the use of descriptions in design, and of developing new design methods that enable users of new or changed computer artifacts to anticipate their future use situation, and to express all their practical competence in designing their future.

This approach is a challenge to rethink traditional understanding of the process of design and its relation to the use of computers in working life. However, it is not only a strategy to include users and their trade union activities in the design process, but more fundamentally to include a cultural and anthropological understanding of human design and use of artifacts, to rethink the dominating objectivistic and rationalistic conception of design. At least in this sense, the collective resource approach reaches beyond the borders of Scandinavia.

THE OBJECT-ORIENTED PERSPECTIVE

The collective resource approach is one part of our background. The other major part - *the object-oriented perspective on programming* - can be traced back to the Norwegian Computing Center as well. Simula67 was initially developed as a simulation language, but its object-oriented approach was found useful as a general programming perspective. Since then there have been major research efforts in Scandinavia within various aspects of object-oriented programming, including also the idea of languages rooted in the professions of the (non-computer professional) users. It is outside the scope of this paper to discuss our view on object-oriented programming further, but we will point out that our way of thinking about programs and programming is strongly influenced by this tradition. Further discussion can be found in [15].

UNDERSTANDING DESIGN AND USE OF COMPUTER ARTIFACTS

Given this background we will now turn to our philosophical understanding of design. We focus on understanding of the role of computer applications in use, on the phenomenology of design, and on design as language-games, rather than on design as consciously planned and executed processes.

An understanding of the role of a computer application in use is important for design. Our inspiration for a new approach to design, based on an understanding of the use of *artifacts* in human work activity, comes from many fields of research. They include the human activity theory of A. N. Leontjew [3], the language-game approach by L. Wittgenstein [10], and recent contributions to the theory of design and computer artifacts by H. and S. Dreyfus [7] and T. Winograd and F. Flores [29].

With these approaches, we take as our point of departure what people *do* with computers in their daily work, how they cooperate with each other by means of computers, and how this cooperation can be enhanced. The basis for design is involved, practical use and understanding, rather than detached reflection. "Hands-on" experiences come into focus. We comprehend design of computer artifacts as concerned social and historical activity in which these artifacts and their use are anticipated. An activity and form of knowledge that is both planned and creative, and that deals with the contradiction between tradition and change.

Design and practical experience

The future use situation is the origin of design, and we design with this situation in mind. To design with the future use activity in mind also means to start out from the present practice of the future users. It is through their experiences that the need for design has arisen, and it is their practice that is to be applied and changed in the future use activity.

Some aspects of practice can be made *explicit.* In design, they can be formally represented in systems descriptions and requirement specifications. But there are other aspects of practice which we can learn only through practical experience. We call these aspects *practical.* The practical aspects are important in design exactly because they are what characterize professional and skillful use of an artifact, as opposed to the use by a novice who basically follows explicit rules.

Design and phenomenology

As mentioned above, our approach is inspired by the one taken by Winograd [29] and Dreyfus [7]. With their phenomenological framework, the point of departure in design is that the different participants understand the situation they come from. They are used to act in situations of "normal resolution". This goes for users as well as designers. The normal resolution or understanding includes the blindness created by the tradition they come from. The design process is characterized by a breakdown of this understanding, by which a situation of irresolution is created. Design is resolving these situations of ir-

resolution, based on commitments between the participants. This is neither objective problem solving nor rationalistic decision making. It is *concerned* human activity, where different traditions and backgrounds meet.

The concept of breakdown is fundamental to design. Breakdown is both desirable and undesirable. On the one hand it is necessary to break down the everyday understanding and use within a specific tradition to create new knowledge and new designs. Breakdown of our understanding of a well known situation is the opening to new knowledge and eventually an understanding of something new. On the other hand, design which is not based on the understanding and use within a tradition – the users' practical skills – are likely to fail, because knowledge "embedded" in the tradition is lost. To be able to deal with this contradiction between involved understanding of the artifact in use and detached reflection on the artifact and the use situation is fundamental to design.

Design as a language-game

Our way of understanding prototyping, mock-ups, and experimental methods in design is also heavily influenced by the ordinary language philosophy of Wittgenstein. Following Wittgenstein, we think of design and use activities as *language-games* that people play: we learn to participate, interact and communicate in games. We use our *ordinary language*, and we acquire competence by learning *in practice*. This means that we view *language as action* rather than *language as description* as fundamental.

Designers are involved in changing computer artifacts and the way people use them. Hence, the language-game of design is one that changes the rules for another language-game – that of use of the artifacts.

Playing the game of design

If designers and users share the same *form of life* it will be possible to overcome the gap between the different language-games. It will at least in principle be possible to develop the practice of design so that there is enough *family resemblance* between a specific language-game of design and the language-games in which the design of the computer artifact is intervening.

The language-games played in design can be viewed both from the point-of-view of the users and of the designers. We can focus on design as a language-game in which the users learn about possibilities and constraints of including new computer artifacts in their ordinary language-games. The designers' practical knowledge will primarily be expressed as the ability to construct specific language-games of design in such a way that the users can develop their reflective and practical knowledge of future use by participating in design processes. However, in order to set up these kinds of language-games the designers have to learn from the users. To possess the competence involved in using a professional language requires a lot of learning within that practice.

The users can, in an involved and influential way, participate in the language-game of design, when the methods applied give their design activities a

family resemblance to the language-games they play in ordinary use situations. In order to stress this important involvement of the users in the design process, we often refer to the users participating in a design process as *lay designers*. They have expertise within the work domain, but no particular expertise as designers.

According to Wittgenstein [30], language-games are also characterized by how we play and make up the rules as we go along. And there are even games where we alter them as we go along. This is in our view a good characterization of the language-games of design.

Descriptions and models in design

In understanding design as language-games, systems descriptions are seen as speech acts we have learned within a specific language-game. If they are good, it is because they are good "moves" within that game. As such they can create breakdowns of understanding as well as help avoid them, depending on what kind of moves they are within the game.

To use descriptions in design is to participate in the playing of a language-game. This is the language-game of anticipating new or changed computer artifacts and use situations. What is created are artifacts that we can *reflect upon*, and some times get "hands-on" involved *practical experience* from (e.g. by using a prototype). Especially artifacts for involved experience as a basis for later reflections are fundamental to our approach.

New design methods?

In summary, it is our position that

- a new design approach must take the specific use activity as its point of departure;
- focus on language as *action* rather than as *description*; and that
- users must be allowed to examine the artifact being designed through hands-on experiences.

What is needed most urgently at the moment is not better linguistic notations for more or less formal descriptions of the functionality of a system, but descriptions that are *reminders of use of the intended computer artifact.* This points in the direction of description methods as support for *concerned involvement*, rather than *correct* description. Such support may be achieved by the use of scenarios, prototypes, mock-ups etc. This is design as a language-game of *doing, learning and playing.*

However, few traditional computer-based design tools are flexible enough to support this kind of design. Traditional prototyping methods exhibit a potential conflict between accessibility (not too much computing competence and programming effort should be needed to use them), and flexibility, both in terms of how the tools can be applied, and in terms of which products can be designed.

With this background and theoretical perspective we now turn to our current research program on computer support in cooperative design and communication.

THE RESEARCH PROGRAM: COMPUTER SUPPORT IN COOPERATIVE DESIGN AND COMMUNICATION

The research program started in May 1987. It is a long term effort planned jointly by the Computer Science Department and the Department of Information and Media Science at Aarhus University [4]. One aspect of the program focus on computer support for experimental design and for communication. The other aspect of the program focuses on the language usage of design and use of computer systems, and the way it relates to the work processes of which it is a part. The purposes are:

- to develop exploratory and object-oriented programming methods into something which, in combination with other design methods, can be applied in practical design;
- to do research into the possibilities of making better user interface design, by means of different theoretical frameworks, and better computer support (such as pluggable software);
- to investigate the possibilities of creating better computer support for cooperative work in small groups.
- to provide empirical knowledge of the interplay between the computer medium and the professional communication that takes place through it, or is motivated by it, and
- to investigate the possibilities for exploiting this knowledge as a basis for design.

As a summary we characterize the theoretical perspective of the program with the following stipulations and reflections:

- *In designing a computer application, conditions for the whole use situation are implicitly or explicitly designed as well.*
  In design of computer support for cooperative work we have to be able to understand the cooperative work the application is to be used for. This can only be done in cooperation with experienced users acting as lay designers.
- *Users and designers often have different backgrounds, different professional languages, and are used to different language-games.*
  The construction of language-games unique to the specific design situation, but with family resemblance with the lay designers normal professional language-games, is an important aspect of cooperative design. In this way, cooperative design becomes a process of mutual learning.
- *Normally, a computer application is used in a multi-lingual environment, comprising the technical support staff and (possibly several) user professions.*

All parties can make legitimate, but sometimes contradictory, demands to the computer application. To design the computer application in such a way that it takes the multi-linguistic environment into account is a challenge in cooperative design.

- *The needs and demands of the prospective users are essential to good design, but are hard to express before the future situation has been experienced.*
  In design of computer support for cooperative work this obstacle can be surmounted by using prototypes, mock-ups, scenarios, etc. which make it possible to get experiences, not only by reflection, but also by involvement in possible future use situations and through use of possible future computer applications.
- *Professional users tend to be rather traditional in their views on how to organize their work and on the potential computer applications for it.*
  Methods in design have to relate to both *tradition and change*, and especially to the interplay between the two positions. Computer applications are often understood metaphorically, and metaphors can be used in design to support the interplay between tradition and change.

We now turn to our own considerations in a project on a computer-based artifact for early envisionment in cooperation between professional designers and lay designers. First we discuss some dimensions of the design situation, we then turn to APLEX, a computer-based environment for cooperative design.

DIMENSIONS OF THE DESIGN SITUATION

As outlined above, we consider the role of the lay designers as a key issue. End-user involvement is needed but to be fruitful, the design situation must have family-resemblance with their work situation and allow them to get "hands on" experiences in situations resembling the (future) work. We call what is demonstrated or examined in the design situation a *prototype*. In doing this we are hopefully not to much in conflict with emerging terminology in the area.

In understanding the design situation we must consider *the people involved.* Are the designers professionals, lay designers, or a combination? Today, the only active designers are professionals. End-users are at best only competent evaluators. Many 4th generation tools advocate that lay designers can design their future computer applications themselves, but this is rarely seen in real projects. Most often we have seen the professional designers as the ones suggesting changes. The users accept or reject, but do not take the initiative to make changes. Furthermore, situations where only lay designers with one type of use background participate differ from situations where more user groups are active. The computer professionals in those situations often take on the responsibility of transferring opinions and choices from one group to the other.

We must also consider different *aspects of the design process.*

One aspect is that of demonstration versus use. In *demonstration* the lay designer watches the professional operating the prototype. By *use* we mean that the lay designer try out the prototype in the (simulated) work activity. In case of *modification* the prototype is changed during a session, whereas in the case of *exploration* the prototype is examined without change. Most practical situations deal only with exploring. This relates to demonstration: a demonstration, which is driven by the professional designers, often resembles a film or video with no possibility of stopping or going behind the screen.

There is also a difference between *laboratory* and *field* (or on location) evaluation: the difference between evaluation of prototypes in an artificial setting and their actual use in the work activity. When we talk about the early stages of the process where envisionment is the main purpose, this is mostly done as laboratory evaluation, or in fact often without considerations for an explicit use setting. Prototyping by versioning can be seen as field evaluation, but at a very late stage of the design process. *Controlled experiment* where the aspects to be tried out are settled in advance differs from *exploratory experiment.* Outside the human factors research, there seems to be little practical or theoretical understanding of the needs or methods for setting up controlled experiments. At the same time, many of the human factors methods are too limited when it comes to the rather complex situations of human work. Furthermore, many approaches remain analytical and do not support design based on the evaluation.

Envisionment may have the character of *brain storm, outline of alternatives or test of a single solution followed by minor changes.* Presently, computer support for brain storming is seldom applied in practice. Often only one basic architecture is prototyped, and then a few different screen layouts, report formats etc. are tested.

Each of these dimensions related to the process are of relevance when creating a cooperative design situation which stimulate active lay designers involvement. For instance, compare a situation where the professional designer in a laboratory demonstrates a prototype with a situation where the lay designer on location tries out various alternatives as part of a brain storming process.

We now turn to some more technical aspects of computer support for design in general.

Depending on the *degree of integration* with the computer resources in the organization, an evaluation in real work situations is made easier or harder. Furthermore, if there is a large degree of integration, all the designers, including lay designers, have a possibility of knowing the computers on beforehand.

Access to other design tools and ways of *combining* various design tools determines the extent to which envisionment have to be done with only one tool or whether it is possible to apply several supplementary tools simultaneously.

Is it possible to reuse and modify modules from existing applications or prototypes? Access to a *component library* helps the designers to rapidly and easily get from one prototype to another. For instance, is it possible to reuse an existing database and build or experiment with different interfaces? To what extent is it possible to experiment with new types of hardware?

The degree of *incrementability* says something about how easily prototypes can be changed: How is it possible to intervene into a prototype in the design session and make the next version running?

Finally, but not of least importance, we draw to the attention the conditions under which the design takes place. The resources available in terms of time for the designers, equipment available and qualifications of the designers are essential. Moreover, the authority of the designers to make decisions about the design process and product is important too.

Based on our theoretical perspective we can conclude that:

- These users should be able to explore and to modify in the field.
- The construction of prototypes should be so effective, and the prototypes so flexible, that different prototypes in fact will be constructed and thus different alternatives tried out.
- The prototypes developed should be based on a suitable spectrum of different computer support. They should be integrated with other systems in the work situation in such a way that the future work situation may be experienced.
- The organizational setting and the resource situation of the designers should allow them to spend the time needed to develop design skills directed towards the specific area of use, and to make decisions based thereon.

This is the design setting for which we need computer support. Unfortunately existing computer support gives rise to systems that are rather closed, with very little support for multi-user applications, for multiple activities, for reuse of existing applications, or for integration with existing applications or newly created applications. Hence, the challenge to design the APLEX.

COMPUTER SUPPORT FOR ENVISIONMENT – APLEX

The design environment APLEX is a means for cooperative design. It is intended to support involved communication among professional and lay designers about future use situations. This communication is based on practical hands-on and organizational experience using APLEX. We see the future design situation using APLEX as being a cooperative design situation between one or more professional designers and one or more lay designers. The different designers are directly involved in the design process and APLEX must be able to respond to the needs of the whole group. This implies that it should be possible for both the professional and for the lay designers to conduct their own ex-

periments using APLEX. Furthermore, it should be possible to engage the designers in intensive design work where the different designers are conducting a mutual experiment using APLEX.

This implies that APLEX must be able to support envisionment ranging from mock-ups over prototyping to application construction/integration, using various techniques such as "intelligent" slide shows, guided tours, and exploratory programming.

On the design of APLEX

APLEX will not primarily aim at *implementing* the computer application which is being constructed. Instead we will experiment with development systems where the prototypes do not need to be running versions of the future computer program, as well as with systems for actual application development. It is one of the aims of this project to develop a designer's workbench, where both possibilities exist as supplementary tools for the designers. The flexibility of APLEX should allow for evaluation of various types of user interfaces, various interaction styles, different functionalities as well as various target applications. APLEX should include generalizations of the facilities that 4th generation tools provide. Furthermore, we will examine the use of various kinds of simulation and visualization techniques. APLEX should also include possibilities of simulating different, specific computer workstations and other types of technology.

Technically, APLEX presents several research challenges. First, it must offer a comprehensive and device independent interface framework. Secondly, it tries to expand the capabilities of prototype systems outside the limits of traditional implementability (parts of the functionality and the interface might be simulated by means of video-disks, "dummy" screen images, or human beings). Thirdly, it explores the capabilities of strongly interdependent interfaces on different workstations connected through high-capacity networks. Fourthly, it explores the relationship between the application and the underlying interface framework through investigation of the technical implications of this concept. To achieve these goal we find that full support of the I^3 concept (*Incremental, Integrated, Interactive*) is necessary. Furthermore, the design of APLEX will be based on many aspects of traditional workstation environments.

Cooperative

From the point-of-view of cooperation, the issue of robustness versus flexibility is important. Often it is the professional designers who need the flexibility whereas it is the lay designers that need the robustness in order to have any realism in their examinations.

In case of demonstration or use in restricted situations where the designers actually sit together and examine the prototype, the "side-tracking" may be avoided by the interference of the professional designer, but in situations where the lay designers are "on their own" this is not the case. Such errors may create breakdowns which cannot be interpreted in the use situation. Hence, they

cannot contribute to the development of the lay designer's understanding of, or unreflected action in this use situation.

Furthermore, cooperation entails that creation of multi-user applications is an important aspect of APLEX, and that even in the design situation multi-person use of APLEX is needed. The multi-user situations create a need to let APLEX include network facilities as well as facilities for sharing of objects.

Multi-person use results furthermore in requirements to documentation and communication support in the design situations. A shared hypermedia is one exiting idea which is, as yet, unexplored. Hypertext technology [6] seems to be an obvious idea for a way of structuring this documentation, because it allows reference pointers among different parts of the text, and even of the prototype. This possibility is primarily intended for reflection in breakdown situations where the "illusion" of being in the future world breaks down for the users.

Incremental

We have described the need for rapid modification of the substrate being created. One important means for achieving the capability for rapid modification is obtained by a high degree of incrementability. Several systems contain such high degree of incrementability, most notable the various Lisp-systems [24] and the Smalltalk-80 system [11]. However, these systems have shown that we face an overall dilemma - the contradiction between flexibility and robustness. Incrementability is obtained by having very flexible programming languages that allow for dynamic binding. At the same time, such dynamic substrates pose serious problems in terms of security. This implies that errors occurring during use are usually indicated at a very low level in the system, which make it very difficult for a lay designer to interpret the actual cause of the error. It should be possible to construct substrates that are consistent, and where errors messages etc. can be interpreted within the substrate.

We also need a powerful debugger (in the line of the Smalltalk-80 debugger) making it possible to cancel erroneous computations, ignore errors, make minor local modifications in order to make progress possible, or follow a chain of activities leading to the erroneous state. The debugger must be able to grant specific capabilities temporarily to an object in order to allow for further examination. We find, however, that this ability must be very explicit in order to ensure that the designer is aware of the change of capabilities of the object. Thus, APLEX should support the manipulation of capabilities of the individual objects.

The above discussion of robustness leads to our view of incrementability. APLEX should be incremental in the sense that it should be possible to modify objects in a substrate without restructuring the whole substrate.

Integrated

By integrated is meant that APLEX is well-integrated with the various other applications in the organization, and that substrates created with APLEX themselves can be integrated with other substrates. The guiding metaphors of

integration in APLEX will be: "access to anything anytime" and "living within the full environment". The metaphor of total access is of course relative to the present capabilities in the system as discussed above. Furthermore, APLEX must keep track of the relations between the objects, and their corresponding source, documentation, help facilities, tutorials, as well as their relations with other objects in the environment.

Having a design tool isolated from the other computer facilities in the organization will give rise to numerous problems. It is therefore important that facilities for connecting APLEX to the existing computer resources are designed to overcome those problems.

The substrates in APLEX will be organized in easy-to-access libraries (or databases) and structured with re-usability and pluggability as some of the most important design strategies. We will extend this view to hardware, such that hardware components in APLEX will be considered similar to software components (pluggable hardware components). Thus allowing for experimentation with alternative hardware devices in the design process and for experiments with advanced hardware, e.g. video.

Besides being well-integrated with itself, APLEX must be well-integrated with other design tools. It should e.g. be possible to use a sequence of screen images made in HyperCard™ [13] or VideoWorks™ [28].

Interactive

Without the need for any arguments, APLEX must contain extensive graphical capabilities for creating highly interactive interfaces both to existing computer facilities, and to APLEX applications. Since we are envisioning APLEX being used for experimenting with the development of applications to be realized on specific computers, we intend, within APLEX, to create simulations of various existing interactive systems, such as the Macintosh desktop [12], the Smalltalk-80 and the Microsoft window systems. This will make it possible, in the design situation, to experiment with the impact of imposing computer-specific constraints on the future application.

Further Design Issues

The above design space gives rise to further important issues, that will be addressed during the design of APLEX. These include:

Domain dependence: APLEX must support application domain specific substrates. This makes it possible to make APLEX "grow" into application domains slowly, and thereby make it possible to create more and more advanced substrates within a specific application domain by creating more and more domain specific substrates.

Enforcement versus conventions: Our main points of reference in the above discussions were Smalltalk-80 on the one hand, and HyperCard on the other. Very alike in some aspects, very different in others. A comparison of the two easily leads to a discussion of what support for programming the prototype designers need – do they need a specific number of different types of objects,

or is a flexible possibility for using and modifying examples of objects better? In general this is a discussion of to what extent a certain style of use of APLEX should be enforced by strict typing and syntax, and to what extent a more flexible guiding of the user by examples, convention patterns, etc., works better. At the moment we do not know, experiences from programming languages are ambiguous in this respect, and we hope to be able to try out different ways of doing this.

Architectural Overview of APLEX

Throughout the development of APLEX we will experiment with object-oriented design. One of the motivations for this is that object-oriented design principles facilitate creation of what can be called pluggable software. That is, software that is open–ended in the sense that a given substrate, created by means of object-oriented design principles, is a substrate that in a future application can be expanded and modified. We will hopefully benefit from this open-ended nature in two ways: First, it will be a well-suited structure for the implementation of the above mentioned components of APLEX, since they both internally and externally are substrates that will be subject to expansion/modification during the design process. Secondly, the strategy for combining substrates will benefit from the use of object-oriented design principles [31]. As such, this part of the project will also be an experiment in realistic application of object-oriented design principles. This leads to the need for a programming language supporting object-oriented design principles. The language chosen is the Beta programming language [16].

The architecture of the graphical interaction system of APLEX contains the following three components:

Graphical library which is primarily concerned with supporting the construction of graphical items. The graphical library is a toolbox with capabilities like drawing (and manipulating) of such items. Presently, APLEX will be designed using the page composition language PostScript as its graphical library [22].

Windowing environment which is primarily concerned with supporting the sharing of the display by various applications running on a workstation, as well as sharing of windows between workstations via a local area network. Within each window, graphical capabilities may be supported, or each application is responsible for utilizing the inside of each window. Presently, APLEX will be designed utilizing the NeWS™ [19] window system.

User interface framework. The design of APLEX is a research effort in the direction of creating a framework by which the interaction between the user and the computer application can be envisioned. The capabilities of such a framework are extensions of the capabilities of the windowing environment. In addition to the capabilities of the windowing environment, the framework contain capabilities for defining more fine-grained structures on the display by defining graphical structures that are not windows, but more tightly connected with the application. Examples of such structures are icons, buttons, menus,

and scroll bars. Furthermore, the framework is concerned with the definition, distribution and handling of events, both hardware events (e.g. mouse movement, keyboard events, etc.) and software events (e.g. window exposed, icon selected, spreadsheet – cell selected). Our view on user interface framework is inspired by the MVC concept in Smalltalk-80 [11]. The semantics of interactive graphical communication are discussed further in [5].

It is important to stress that our view on user interface frameworks is not part of a discussion in favor of separating the design of the interface from the design of the functionality of the application. In fact, we find that such separation is neither possible nor feasible in general [27]. However, it is our aim to create a set-up of pluggable components, some of which deal with the interaction and some with the *underlying components*, e.g. databases.

Some applications may be constructed with more than one interface associated with it. There are several architectural reasons for this: Each interface may, for instance, focus on specific aspects of the application, and the structure of each interface is designed in order to ease the manipulation of these specific aspects. These interfaces might all be active at the same time, and manipulations of the application through one of the interfaces might influence the other interfaces. Each interface defines a protocol that the application must support, and the interfaces must be dynamically connected to the underlying components. Furthermore, interfaces may utilize various predefined interface components, such as buttons, scroll bars, or menus. Such a framework will allow for rapid modifications of a prototype, as well as for design of different alternative prototypes, some of which show different styles of interaction based on the same underlying components.

The above structure is what we, with respect to user interfaces, mean by pluggable components. The protocols define the *slots* by which interfaces and underlying components can be *plugged* together. In the design of APLEX, we will examine the usage of object-oriented design principles in this area.

The underlying extensibility in object-oriented systems seems to be well suited to pluggability. The design of APLEX will utilize this pluggability as the fundamental architecture of the system. This implies that all components of APLEX will be constructed as objects, including hardware components. In this way, we will be able to simulate not yet constructed specialized hardware by constructing a software simulation (software object) of that component and conduct experiments. Furthermore, we will be able to experiment with different hardware solutions to specific interaction problems by defining common properties of types of hardware (e.g. pointing devices or picking devices) and then select different actual devices (e.g. soft screen vs. mouse vs. tracker ball). In the same way, it is possible to encapsulate the functional behavior of external (to APLEX) software systems as objects in APLEX with a protocol, modeling the functionality. Furthermore, we will be able to treat external resources on equal terms with APLEX resources and experiment with using different external resources as alternatives in a design process.

Using APLEX

We would like to conclude our treatment of APLEX with a "Please try it!". At the moment, however, APLEX exists only as envisionment. We are now conducting experiments, based on HyperCard, Smalltalk-80, NeWS and other systems in order to try out and look further into the ideas outlined above. We have also initiated the construction of the first prototype of APLEX, while still continuing to develop the conceptual framework underlying APLEX.

As a weak substitute for "hands-on" experience we make use of a fictious example. The example is, however, firmly rooted in our empirical research [3].

Imagine a project where a group of professional designers work together with a group of office workers in a government institution to help these office workers achieve new kinds of computer support for their work. The project is managed by a steering committee with representation from management and the employees. It is a basic idea of the project that the employees should, in project groups, take part in designing the computer applications that they are to use themselves. The specific case deals with the filing and retrieval system for incoming and outgoing mail etc., the so-called Journal. The purpose of this project is to find out how the Journal can be reorganized to be more efficient, eventually by means of a computer application.

From the beginning, the group work with three different alternatives:

- a restructuring of the existing paper based Journal without the use of computers.
- a restructuring of the paper files with computer support for retrieval of documents and computer-based mail lists to inform the workers who draw on the services of the journal in their daily work (case workers) about incoming mail.
- a computer based Journal where all documents are scanned in upon arrival in the Journal office, and with computer based retrieval and mail lists.

In the early meetings it is a major task to delimit the type of computer application wanted from the three general solutions. Some of the important issues are the organization of work – who should use the application and how?, what are the hardware choices?, and how are they connected to the physical organization of work? Depending on whether the documents are to be filed in a traditional paper file or scanned into the computer, the women in the Journal office have to conduct their work differently – the role of e.g. a photo copier would differ. Much of the internal mail circulation would not be needed with the scanner solution, i.e. the traditional communication patterns would be changed fundamentally.

In this situation, the professional designer initiate the discussions by building two small *demonstration* prototypes by means of APLEX: scanner or no scanner. Together with simple mock-up's supplementing the physical layout of the future work-place, the prototypes are used to demonstrate to the group what the possibilities and constraints of the *alternatives* would be. Maybe APLEX

doesn't really contain a pluggable scanner, in which case the designer uses images on a videodisk, made with a drawing program, just simulating the scanning procedure. The main idea is to get a discussion about technical and organizational implications of the two different proposals.

In this situation, APLEX is a flexible environment for the professional designers: it allows them to reuse parts from one prototype in constructing the other; to make use of other design tools in building the prototypes, and to use pluggable hardware devices.

Next we consider a situation where a professional designer and a group of lay designers, women from the Journal office, sit down to find out exactly what information should be filed, how it should be entered, what the screen images should look like, and what interaction they want. Based on earlier talks with the users and on the previous meetings, the professional designer has made a first prototype. This prototype is merely a sequence of screen images, which are based on the appearance of the mail lists presently used at the institution. When necessary, information is added to the prototype. The discussion focuses on the information needed – on the screen and in the files, and how much of this should be entered by the office workers – and on the possible changes in ways of cooperating. This is a situation where the designers are *modifying* the prototype, simultaneously with the *use* of it. The component library is used to look at different types of screen layouts and interaction styles: a direct manipulation version, a form-filling one, etc.

At a later state, the prototype, which have been elaborated on by the involved group, can be used in its *real organizational setting*. The prototype is still running by means of APLEX, but now APLEX needs also to be hooked up to, or running some of the other computer programs that are normally used in the Journal office, e.g. a word processing program. For a period of time, the Journal office tries out their design in their daily work. Some changes are made in the way the system is used. Problems still come up about the information needed to file and retrieve the documents, but also about the speed of the application. The situation is one where the *robustness* of the prototype is important since the professional designers, although present, cannot help each user all the time.

SUMMING UP

We have seen examples of the use of APLEX in different *situations* in a design process. This process is one in which the participants could make use of their different backgrounds as office workers or as designers in playing the language-game of design. The APLEX prototypes have made it possible, under different conditions, for the office workers to experience their future work situation. The APLEX is an environment that facilitates such language-games. We do not see APLEX as the only way of doing this. Rather, APLEX ought to be one of many more or less integrated tools and techniques belonging to the practice of the professional designer.

The design process is very important for the future work situation of the users. The kind of computer support needed for cooperative work in different settings may differ a lot, and as we have argued, it is important to investigate and develop new possibilities of cooperation in a design process where prospective users are actively involved. In other words, we do not primarily see cooperative work, or computer support for it, as a static entity. We view design as a cooperative process out of which new possibilities of cooperation is created.

In the presentation of our example we have made a number of gross simplifications, especially with respect to the degree of harmony in the project: first of all, management doesn't interfere with the process. Secondly, there is only one user group. From the real life case we have reduced complexity by not considering the group of case workers. Thirdly, we assume that the professional designers have no interests of their own, which contradicts those of the workers. Real design processes are surrounded by many conflicting interests: the conflicts between management and labor, workers collectives which question why they should do a job that they are not hired to do: help management design computer applications, and conflicts among groups of workers who belong to different trades. Another simplification, closely related to our neglect of conflicts, concerns the rationality of the cooperation in the design process described: it does not differ significantly from the small research group ideal discussed in the beginning of our paper.

To get beyond the small research group ideal and reach a fuller understanding of what cooperation means in real life situations is a major challenge for our research program. Not least because design is a process of change in which the tools and materials of a group are often replaced by something new. If we restrict ourselves to the shared material, shared tools, etc. definition, we cannot understand how groups cope with situations of change, such as design, when their traditional "sharedness" - the tools and materials - are taken away from them. Such a group needs not only cooperative work as an ideal, but as a (design) process leading in a democratic direction.

REFERENCES

1. Bjerknes, G. et al. (eds.): *Computers and Democracy – a Scandinavian Challenge*, Avebury 1987.
2. Bødker, S. et al.: A Utopian Experience, in [1].
3. Bødker, S.: *Through the Interface – a Human Activity Approach to User Interface Design*, DAIMI PB-224, Computer Science Department, University of Aarhus, 1987.
4. Bøgh Andersen, P. et al.: *Research Programme on Computer Support in Cooperative Design and Communication*, DAIMI IR-70, Computer Science Department, University of Aarhus, 1987.

5. Bøgh Andersen, P., Knudsen, J.L.: *Semantics for Interactive Graphical Systems*, Preliminary version, Computer Science Department, Aarhus University, 1988.
6. Conklin, J.: *Hypertext: An Introduction and Survey,* IEEE Computer, 20(9), September 1987.
7. Dreyfus, H. L., Dreyfus, S. D.: *Mind over Machine – the power of human intuition and expertise in the era of the computer*, Basil Blackwell, 1986.
8. Ehn, P., Sandberg, Å.: Local Union Influence on Technology and Work Organization, some results from the Demos Project, in Briefs, U. et al. (eds.): *System Design, for, with, and by the user*, North-Holland, 1983.
9. Ehn, P., Kyng, M.: *The Collective Resource Approach to Systems Design* in [1].
10. Ehn, P.: *Work-Oriented Design of Computer Artifacts*, Almqvist & Wiksell International, Falköping, 1988.
11. Goldberg, A., Robson, D.: *Smalltalk-80: The Language and its Implementation*, Addison-Wesley Publishing Company, 1985.
12. *Human Interface Guidelines: The Apple Desktop Interface*, Addison-Wesley Publishing Company, 1987.
13. *HyperCard User's Guide*, Apple Computer, 1987.
14. Johnson, B., Weaver G.: Using a Computer-Based Tool to Support Collaboration: A Field Experiment, in [23].
15. Knudsen, J.L., Madsen, O.L.: Teaching Object-Oriented Programming is more that Teaching Object-Oriented Programming Languages, in *Proceedings of Second European Conference on Object-Oriented Programming (ECOOP`88)*, Oslo, Norway, August 1988.
16. Kristensen, B.B. et al.: The BETA Programming Language, in Shriver, B., Wegner, P. (eds.): *Research Directions in Object-Oriented Programming*, MIT Press, 1987.
17. Kyng, M., Mathiassen, L.: Systems Development and Trade Union Activities, in Bjørn-Andersen, N. (ed.): *Information Society, for Richer, for Poorer*, North-Holland, 1982.
18. Lysgaard, S.: *Arbeiderkollektivet*, Universitetsforlaget, Oslo 1976 (In Norwegian).
19. *NeWS Technical Overview*, Sun Technical Report, 800-1498-05, 1987.
20. Noddings, N.: *Caring. A Feminine Approach To Ethics & Moral Education*, University of California Press, 1984.
21. Polanyi, M.: *Personal Knowledge*, Rutledge & Kegan Paul, 1967.
22. *PostScript Language Reference Manual*, Addison-Wesley Publishing Company, 1985.
23. *Proceedings of the first Conference on Computer-Supported Cooperative Work*, Austin, Texas, December 1986.
24. Shiel, B.: *Power Tools for Programmers*, Datamation, 29(2), February 1983.

25. Stasz, C., Bikson, T.: Computer-Supported Cooperative Work: Examples and Issues in One Federal Agency, in [23].
26. Sørgaard, P.: A cooperative work perspective on use and development of computer artifacts, Järvinen, P. (ed.): *Proceedings of the 10th Information Systems Research Seminar in Scandinavia*, Tampere, 1987.
27. Tanner, P., Buxton, W.: Some Issues in Future User Interface Management System (UIMS) Development in Pfaff, G.(ed.): *User Interface Management Systems*, Springer Verlag 1985..
28. *VideoWorks II Manual*, MacroMind, Inc., 1987.
29. Winograd, T., Flores, C. F.: *Understanding Computers and Cognition: A New Foundation for Design*, Ablex Publishing Compagny, 1986.
30. Wittgenstein, L.: *Philosophical Investigations*, Oxford University Press, 1953.
31. Yankelovich, N. et al.: *Intermedia: The Concept and the Construction of a Seamless Information Environment*, IEEE Computer, 21(1), January 1988.