Supporting Active User Involvement in Prototyping[†]

by

Kaj Grønbæk
Department of Computer Science
University of Aarhus, Denmark
Email: kgronbak@daimi.au.dk

Abstract

The term *prototyping* has in recent years become a buzzword in both research and practice of system design due to a number of claimed advantages of prototyping techniques over traditional specification techniques. In particular it is often stated that prototyping facilitates the users' involvement in the development process. But prototyping does not automatically imply active user involvement! Thus a *cooperative prototyping* approach aiming at involving users actively and creatively in system design is proposed in this paper. The key point of the approach is to involve users in activities that closely couple *development* of prototypes to early *evaluation* of prototypes in envisioned use situations. Having users involved in such activities creates new requirements for tool support. Tools that support *direct manipulation* of prototypes and simulation of behaviour have shown promise for cooperative prototyping activities. Examples of such tools are given and the ways that they support cooperative prototyping within various use domains are discussed.

Keywords: System Design, User Involvement, Cooperative Prototyping, Direct Manipulation, Simulation, Prototyping Tools.

[†] Grønbæk, K. (1990). Supporting Active User Involvement in Prototyping. *Scandinavian Journal of Information Systems* 2:(pp. 3-24). The paper is a revised version of the paper: Grønbæk, K. Extending the Boundaries of Prototyping - Towards Cooperative Prototyping. *In the Proceedings of the 12th IRIS conference*, DAIMI PB 296-I Aarhus University, Denmark, December 1989.

1. Introduction

Recently, literature on prototyping methods and techniques has emerged, and it is slowly replacing the literature on structured methods of the 70's. This is illustrated by book titles such as ``The Prototyping Methodology" (Lantz 1986), ``Application Prototyping - A requirements definition strategy for the 80's" (Boar 1984), and ``Software Prototyping, Formal Methods and VDM" (Hekmatpour & Ince 1988). Discovering this trend one could ask the question: How does this methodological change towards prototyping really improve the users' opportunities to be involved in and to influence system development in their organizations? My educational background is in the branch of Scandinavian system development traditions, described as the *critical* school by Bansler (1989). A key point in this tradition is to take the users' practice as the starting point for new development. Given this background, I previously engaged in empirical studies of the use of prototyping techniques in IS development projects (Grønbæk 1989) to study the question posed above. These studies showed that even though the developers used prototyping approaches similar to those recommended in the literature, the users were only brought in to evaluate prototypes based on demonstrations. These prototyping activities did not result in significantly better system designs than did earlier description methods. A number of overlooked requirements were found when the users experienced the new system during test and installation activities.

These observations prompted an investigation of how prototyping techniques and tools could be improved to enable more active and even creative involvement of users in system development. Elements of this investigation have been carried out by studying the use of a so-called *cooperative prototyping* approach in various use domains - examples are described in (Bødker & Grønbæk 1989 and 1990). The research is carried out within the research program ``Computer Support for Cooperative Design and Communication", described in (Bøgh Andersen 1987, Bødker et al. 1987). In this program we are, among other things, working empirically in different use domains and experimentally in the laboratory to develop tools and techniques to support the kind of cooperative design argued for here. This paper introduces the cooperative prototyping ideas and provide a survey of potential tools to support cooperative prototyping. The structure of the paper is as follows: Section 2 gives a motivation for cooperative prototyping. Section 3 summarizes the ideas behind cooperative prototyping as they have developed through the field studies mentioned above. Finally, section 4 gives a brief survey of the technical possibilities for supporting cooperative prototyping. This survey is meant to identify prototyping tools that address the goal of having users involved in aspects of prototype development as well as in work-like evaluation of the prototypes.

2. Challenging current prototyping approaches

A rich variety of approaches to prototyping have been proposed in recent years, as mentioned in the introduction. They all provide possibilities for users to gain hands-on experience before the final application is built, and yet, they often fail to be applied this way. In the literature roughly three categories of prototyping approaches can be identified: ``the prototype becomes the system" approaches, executable specification approaches, and exploratory approaches. The *``prototype becomes the system"* approaches (Boar 1984, Boehm 1984, Canning 1981, Lantz 1986) dominate current systems development practice and literature. They aim to supplement a traditional requirements specification with a prototype of the user interface aspects before the implementation begins, primarily for the users to adjust details of the system. Alternative design proposals are decided upon without development of alternative prototypes. Empirical studies (Grønbæk 1989) show that such prototypes are used primarily for passive demonstrations of features, and not to let users try features out actively. The main purpose of the *executable specification* approaches (Hekmatpour & Ince 1988, Squires et al. 1982) is to obtain a full, formal specification of (parts of) the future system behaviour. The specifications are made in an executable specification language, meaning that a program, and thus a prototype, can be generated automatically from the specification. In theory, users and designers can evaluate the specification by evaluating this prototype; in practice, this is rarely ever done. The specification languages are usually not suitable as a means of communicating with users, and a full-scale formal specification of a system has proven to require an effort similar to traditional programming of the system. Offsprings of these approaches are seen in the so-called CASE-tools[1]. However, code generating CASE-tools have not yet been used enough to assess their ability to support prototyping. In *exploratory approaches* (Bødker et al. 1987, Good et al. 1984, Hsia & Yaung 1988, Mogensen 1985, Thomsen 1987) mock-ups, simulations, and ``throw-away" prototypes are developed employing various tools. The goal is to use quick-and-dirty sketches of the computer application to clarify requirements for a new computer system. A number of cases describe user involvement in evaluation of prototypes, but few examples in which users have been actively involved in the design and modification of prototypes, and thus creatively influencing the future system. However, the rapid development of simulated applications does facilitate early hands-on evaluation. The prototypes in these examples serve mainly as a means for idea exploration before they are turned into a specification of the application to be developed.

The first two categories of prototyping approaches assume a pure software engineer, manager, or designer perspective on system development; i.e., the goal is to develop reliable programs at a low price. Some of the exploratory approaches, however, attempt to consider a user's perspective. In these approaches, the usability of the systems is considered to be at least as important as, program correctness and low development costs. Taking a user's perspective would be to focus on work processes, division of labour, skills required for new work tasks, what interaction with the new system will be like, etc. A user perspective would thus be to require usable and tailored computer systems that support their work in a better way. Described with the concepts introduced in (Winograd & Flores 1986) and (Bødker 1987a) a high quality computer system is a system that enable users to improve their work and perform it with a minimum of breakdowns caused by the use of the system. The system should be as transparent a tool as possible for the user. Program correctness is of course a prerequisite for minimizing breakdowns, and low development costs are also important to users who want a new system. But these issues are usually not the main concerns when users evaluate a new system design; the focus is rather on what future work with the system will be like.

In developing high quality systems that meet the users' expectations, current approaches to prototyping share a number of drawbacks with traditional system description approaches, particularly in the case of the most widely utilized ``prototype becomes the system'' approaches. This is due to the typical restriction of users to have a quite passive role in the design process. The users act as evaluators of technical proposals, made by designers, rather than as active contributors to the design of the computer system that will influence their work-place. The users do not have sufficient opportunity to anticipate the potential that breakdowns the proposed system can cause, or to feel how the proposed system will change their workplace and possibly the whole organization.

Increasing user demands on high quality computer systems

To conclude, passive involvement of users is insufficient to meet the challenges of both current and future development of computer systems. A number of factors contribute to increased user demands for access to computer systems that are tailored to the particular requirements of an application domain. First, greater use of computers in organizations and private homes increases the familiarity with computers and different styles of interaction. Thus, we may expect higher user expectations regarding the design of computer systems. Second, people are becoming more dependent on computer support for the tasks they are performing in their daily work. Most users in the future have experienced or heard about bad systems. Hence we may expect users to become more interested and demanding when (if) getting the opportunity to influence design of a new computer system. Third, a number of computer applications in corporate businesses, such as budgeting, database maintenance and retrieval, statistics, and reporting, are now widely supported through the acquisition of standardized program packages. Some program packages allow users to tailor the system to meet their needs. Thus, requests for in-house designers or consultants will more frequently occur in new application domains, where goals often are often less certain. Finally, the technological innovation as represented by networked graphical workstations or PC's provides new opportunities for the design of computer systems. In particular new interaction styles may provide greater latitude for supporting the variation and complexity in users' work. Hence technological innovation itself creates further challenges for system designers. The number of alternative interaction styles to choose among in the design process increases dramatically. This increase also adds to the importance of experimenting with different design ideas and envisioning the alternatives work-like situations.

Trends showing increasing user demands in system development organizations are described in (Friedman & Cornford 1987) based on an international empirical study. These trends challenge system developers to improve their work practices. Some of these challenges may be met by technical advances. But an important challenge is to involve users actively and creatively throughout the development process, to improve the usability of future systems. This involvement is crucial to ensure that new computer systems are tailored to the users' needs and support the development of their skills. Section 3 outlines ideas on how to meet the challenges of user involvement by changing our view of prototyping.

3. Active user involvement in prototyping

Taking the perspective that the goal of system development is to create well-tailored systems for users, I propose a different view of prototyping inspired by Bødker (1987b). Design, including prototyping, should be a process in which designers and users in cooperation determine and create the conditions for the users' future work. To explore how a future computer system can improve the users' work situation, the users must be able to somehow ``experience'' the future system in use. To experience a future use situation is *not* to read a description of a computer system or of its use, *nor* is it to watch a demonstration of a prototype: Human beings possess skills that are not easily articulated or imagined detached from a situation of acting. For example, humans normally use tools in the work process without being aware of them. Skills of using the tools are usually not articulated, they are triggered in the work situation when dealing with a certain piece of material. To make such tacit skills contribute to the design of a computer system, the users have to actually try out the future system in a situation of use or imagined use. This argument can be supported by the observation in (Grønbæk 1989) that a lot of criticism of new computer systems occurs when, during the completing phases of development, they are tested by a group of actual users: A criticism which occurs even though the users' had the opportunity to watch demonstrations of prototypes earlier in the process.

Thus, good design may require setting up situations in which it is possible for the users to gain hands-on experience with the future system before a design proposal is frozen. Instead of developing a full-scale system to provide such hands-on experience, tangible models, such as mock-ups, simulations, or prototypes, can be used early in the development process. Using the term ``tangible model'', I want to emphasize that it should be both a working model that affords some use and is flexible enough to be immediately manipulated to some extent (see section 3.2 for a more detailed discussion). While ``in the future'' conducting a future work task using a tangible model, certain breakdowns will cause the fluent conduction of the work task to stop. For example, the users may become consciously aware of the tool that they are using, because it does not fulfil its purpose. In a prototyping process such a breakdown may motivate a change of the prototype, and eventually a change of the overall design of the future computer system. Refer to (Winograd & Flores 1986, Bødker 1987b) for a discussion of the breakdown concept.

The widely applied approaches to prototyping also overlook the importance of considering alternatives during prototyping activities. Hence, the approaches create a *blindness* of design very similar to traditional structured description methods. See (Winograd & Flores 1986) for a discussion of blindness in design. In a prototyping process, alternative prototypes should ideally be considered visionary discussions in which both users and designers can pose ``What if.....?'' questions on a more concrete basis. Consideration of alternative design proposals is in general valuable, because the confrontation between contrasting and perhaps conflicting ideas stimulates innovation. The need for consideration of alternatives has been argued elsewhere in (Grønbæk 1989).

The proposed goals for future development of prototyping tools and techniques are: To enable users and designers to envision future work situations with prototypes, and to provide support for flexible exploration of alternatives early in the development process. Hence, system designers need to change their view of prototyping. Prototyping needs to go beyond having a few users watch a demonstration and approve an early, prototype version of ``the system". Combinations of prototyping approaches having their roots in the exploratory prototyping approaches described in section 2 need to get more attention in general, particularly in innovative development projects, where ideas are fuzzy and users request a tailored system rather than a standard system. But to benefit fully from the exploratory prototyping approaches it is still necessary to allow the users a more active and creative role. From the designer perspective this is required because the users' skills are crucial for development of a tailored system that fits their needs. From the user perspective this is required, to get a feel for how new technical possibilities can or cannot support their work, and to get the opportunity to *influence* the development of computer support for their work.

3.1 Cooperative prototyping

To meet the goals outlined above I propose a prototyping approach, labelled *cooperative prototyping*. The aim of the approach is to establish a design process where users are involved *actively* and *creatively* in prototyping activities cooperating with designers. The idea of the approach is to more closely couple analysis and design activities by rapid development of one or more tangible models early in the development process. The initial models should help make the participants' visions concrete. In order to be concrete they need to relate to core work tasks. These possibly alternative tangible models can then be modified, thrown away, or built anew in an iterative process that increases the participants' understanding of technological possibilities related to the users' work tasks. The activities may serve one or both of these goals: 1) Idea generation and exploration or 2) work-like evaluation of the models. When serving 1) the focus is on cooperative use of the prototyping tool and prepared building blocks to create a tangible model or an extension of an existing one. When serving 2) a fairly robust model has been prepared. The users get some training in its use and then an evaluation based on work-like testing of the model is undertaken. Breakdowns in this testing, caused by faulty design lead to immediate modifications of the models. Breakdowns due to other causes are handled otherwise; for instance, lack of training is handled by further training.

A cooperative prototyping process can ideally continue until the group decides that they have a sufficiently clear idea of how the features of the model should be represented in the final system. The designers also need to have an idea of how it can be implemented. Cooperative prototyping need, however, not to be limited to early stages of system development. The approach can be valuable throughout a project in particular with respect to establishing work-like evaluation. It is also worth noting that all of the prototypes developed with this approach need not be thrown away when implementation begins. It would be an advantage to be able to reuse some of the screen objects that has been developed. But to get a cooperative prototyping process going it is important to be able to create and modify quick-and-dirty functionality without consideration of performance issues etc. Thus, functionality often needs to be programmed anew or cleaned up during implementation following cooperative prototyping.

Cooperative prototyping processes are performed by a group of designers in cooperation with a group of users. Cooperative prototyping requires that the group of designers and users have access to tools that support rapid development and modification of tangible models. The models should provide enough (simulated) functionality to make it possible to experience the dynamics of future work tasks. This requirement determines a need to go beyond wood and cardboard mock-ups and develop computer based tools to support the process. Modifications should be made rapidly either on-line in combined evaluation/design sessions if appropriate, or immediately after the session enabling a new evaluation/exploration ``the day after". This is to emphasize that the users should not have to wait months or years to experience enhancements of the prototype.

Examples

Bødker & Grønbæk (1989) describe a cooperative prototyping process that focuses on evaluation in work-like settings. HyperCard™ - a flexible and cheap design tool available on Apple Macintosh computers - was used by designers working in close cooperation with a group of dental assistants to prototype a dental case record system. This was done in a series of cooperative prototyping sessions in which 2-3 users and a designer used an Apple Macintosh workstation. We utilized the direct manipulation facilities of HyperCard to make in-session modifications when breakdowns occurred in simulated work situations. Breakdowns due to design problems were discussed and either turned into an immediate modification of the prototype, a proposal for later change, or further instruction on how to use the prototype. The prototype simulated connections to remote databases. The inherent functions of HyperCard made it easy to experiment on alternative search strategies and ways of using a display of representation of the human set of teeth. Finally, two alternative ways of representing a treatment of teeth in the case record were explored. We learned from these prototyping sessions that a reasonable small effort (a week's work for a designer) was enough to prepare tangible computer based models that could be used both as means for establishing a work-like situation, and as a means to be extended and modified interactively within the prototyping sessions.

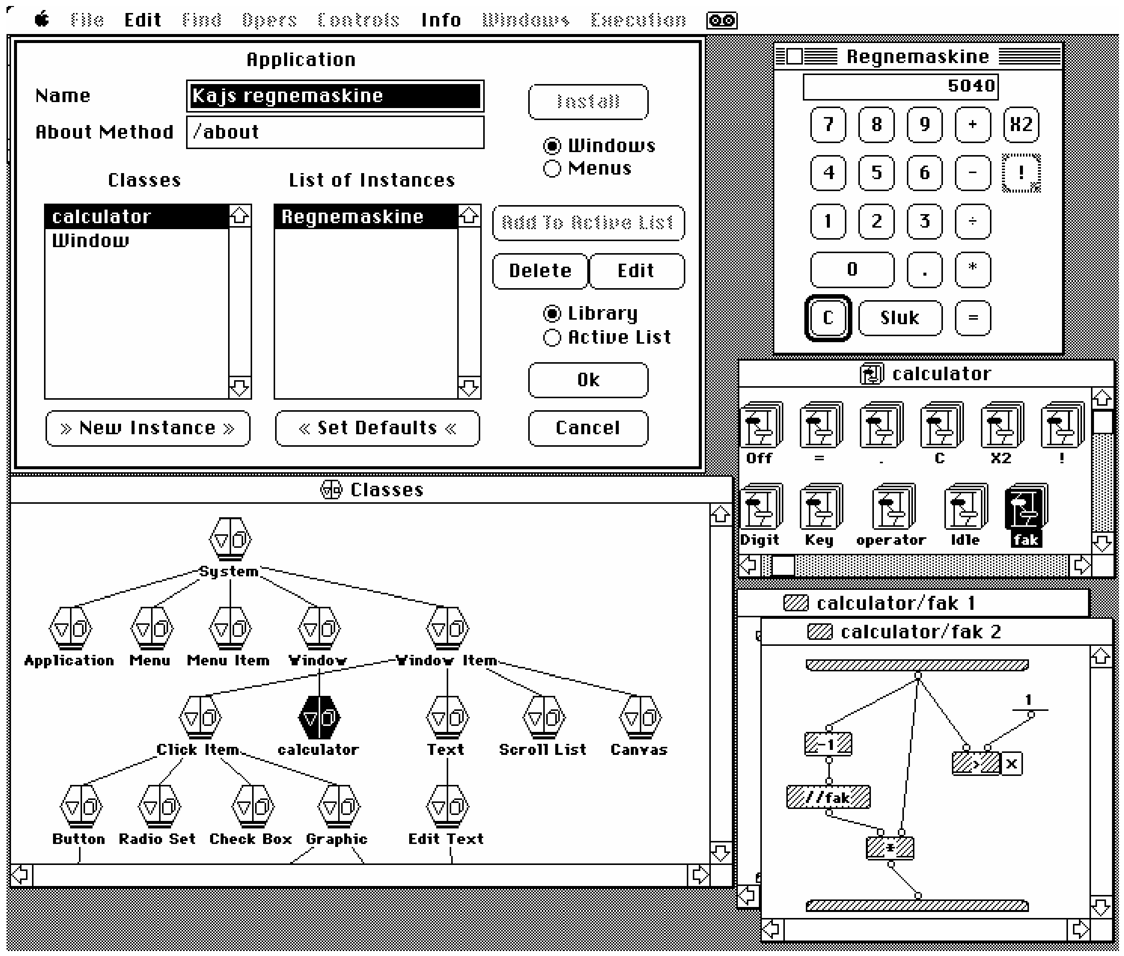
In (Bødker & Grønbæk 1990), similar prototyping sessions were undertaken with a group of caseworkers, i.e. architects, engineers and drafts(wo)men, from a technical department in a Danish municipality. A technical department is responsible for urban planning and environmental control among other things. We were designing integrated support for casework in the technical department. For this purpose, maps, large texts, and common databases were combined in hypermedia-like prototypes. Hypercard was used to build parts of the prototype and to integrate other applications. Based on this project, (Bødker & Grønbæk 1990) document experiences on using cooperative prototyping for idea generation and exploration

A third example that focussed on work-like evaluation is (Hauerslev & Jakobsen 1988). Prototyping sessions aimed at designing a telephone switchboard for secretaries were undertaken in close cooperation with a user, utilizing a domain specific programming tool built on top of a programmer's interface to a network supported graphics package called NeWS running on SUN workstations. In this case a user and a designer evaluated a prototype running on a graphical workstation. Needs for modifications discovered when breakdowns occurred were made on-line by a programmer working on another connected workstation in the same room.

These examples are prototypical examples of cooperative prototyping processes. They indicate that it is becoming realistic to perform cooperative prototyping early in development projects. There is no reason to believe that the approach causes economic trade-offs. To begin with the approach leads to development of better quality computer systems, i.e. systems tailored to the users needs. It is quite common in projects to discover new requirements during installation activities, because systems do not meet the users' expectations. Such requirements may be anticipated through prototyping experiments early in the development process, thus saving resources at the end, to the benefit of the project as a whole. Boehm (1981, p. 39 ff.) claims that ``errors'' detected in the late phases of a development project can be up to 100 times more expensive to correct than the same errors detected in the requirements determination phase. In addition the potential sources of computer support are emerging. Section 4 supports this by giving examples of current tools that can be used either to support cooperative prototyping or as platforms to build such tools.

3.2 Tangibility and direct manipulation

Cooperative prototyping combines the use of computer-based tools for exploratory prototyping with approaches to design that allow users to participate in the modification of wood and paper mock-ups. To undertake such a process it is important that users and designers are able to work closely together with a minimum of communication problems caused by the prototyping tools themselves. The tools should ideally be understandable, tangible, and transparent[2] to both users and designers, as is the case with the use of paper and scissors to make mock-ups (Bødker et al. 1987). The term tangible models was introduced earlier in this section to describe means for design that can be manipulated with such tools. Various combinations of manual, computer based, and even interactive video-based models may be used as tangible models for cooperative prototyping. Users and designers can quickly sketch out and explore ideas with them. To allow evaluation in work-like settings, we need functionality behind the tangible models, enabling evaluation in simulated use situations as a supplement to the flexible sketching of ideas. The computer based kernel can provide system behaviour that allow hands-on testing of behaviour aspects such as navigation between facilities, entering data, and finding requested data. But the tools should still be flexible in the design situation and ideally understandable to the whole group, as with wood and paper mock-ups. This situation is still hard to realize with computer-based tools; for instance, a programming language is not easily understood by non-programmer users and thus won't support user creativity.



Name: Kajs regnemaskine

About Method: /about

[✓ Windows] Menus

[✓ Library] Active List

Figure 1: A screen showing use of a direct manipulation style design tool, Prograph for Apple Macintosh. The window for an application, a calculator, is being extended with a new button to calculate factorial. An iconic representation of the program to do the calculation is shown in a window below the application window.

However, tools that provide support for *direct manipulation*, i.e. point-and-select interfaces for manipulation of objects in a computer system, are emerging. See (Shneiderman 1983) for an introduction to the direct manipulation concept. In cooperative prototyping situations involving users, direct manipulation tools have a number of intriguing advantages compared to conventional programming tools. Manipulation of objects become visible, because of immediate feedback that highlights the object to be manipulated. Similarly, the result of an operation is immediately made visible when possible. Moreover, operations on objects are often chosen from menus and dialogues. In such interaction, longer and more meaningful labels can be used, because it is not necessary to type in the labels to perform the operation as is the case with conventional programming and command oriented interaction.

The interaction is characterized by selecting objects and tools, and by using selected tools on selected objects (see Figure 1 for an example). Such interaction resembles other kinds of human work and makes the design process easier for users to follow. Design tools providing such direct manipulation interaction can reduce the asymmetry between users' and designers' understanding of the design process. However, it will still be the designer who possesses the skills needed to understand the design tool in detail. Thus, the designer will possess the power when outlining technical possibilities. Such tools are suitable for developing computer-based tangible models that both provide functionality to be tried out and the ability to be manipulated directly in an iterative process of cooperative prototyping.

Mock-ups, simulations, and prototypes are kinds of tangible models that have been used in exploratory prototyping, as described in section 2. But in IS development practice, exploratory prototyping approaches are considered inefficient because the designers have to throw away (parts of) prototypes. Thus, designers only develop prototypes that are suitable to ``become the system''. This kind of prototype development is supported by the widely spread 4th Generation Systems and Application Generators. These tools are usually limited to support direct manipulation of form-based dialogues and report lay-outs, and their ability to support simulation of functionality is quite limited. Section 4 describes tools to support development of a much wider range of tangible models that allow simulation of behaviour and direct manipulation.

4. Support for cooperative prototyping

This section briefly surveys sources for developing computer support for cooperative prototyping. Bødker et al. (1988) identifies a set of requirements for tools to support cooperative design based on a discussion of properties of cooperative design situations. The importance of having access to Interactive, Integrated, and Incremental tools is highlighted. Moreover, they argue that the tools should ideally be provided as domain specific substrates[3] embedded in a general Object Oriented development environment. I agree with those general requirements, but from the discussion above and practical experiences with cooperative prototyping (Bødker & Grønbæk 1989 and 1990) I find it important to highlight the requirements for direct manipulation and simulation support. Direct manipulation facilities help in involving users in building and modifying prototypes and the simulation of functionality help in setting up work-like evaluations. A number of different computer-based tools that could support cooperative prototyping are discussed briefly. The tools are grouped according to the techniques they support , as follows: Animation, Simulation, Wizard of Oz, Design by Example, and Visual Programming.

4.1 Animation

Several tools are capable of supporting design of screen objects such as buttons, windows, and form-based dialogues by allowing users to draw figures, write text, and place fields and icons on an empty screen or a window. To provide functionality behind such screen objects, a switch to another mode and environment may be required. A larger programming effort is also involved. However, behavioural aspects such as making connections and moving among various screen objects can be illustrated with animation facilities.

When dealing with purely form-based dialogues for character-oriented screens, screen editors in 4th Generation Systems such as SQL-FORMS of ORACLE[4] provide a step-by-step animation of a sequence of dialogues; for instance, by using function keys to trigger transitions between prepared screen layouts. Data-items can be entered in the fields of the screen layouts, and are held by the program until the end of the session. Screen layouts constructed early in the design process can through iterative modifications be used later during the implementation of the system. In (Hsia & Yaung 1988) a dedicated screen-editor and animation system is presented, which supports data entry and the specification of simple branching conditions for the animation of a sequence of dialogues. In this case, however, the screens and their sequences, here denoted a scenario, are not integrated in a system that allows reuse for implementation.

Tools for graphical user interfaces that include windows, buttons, dialogue boxes, pop-up menus, etc., may provide similar animation facilities. One possibility is to use tools such as VideoWorks and HyperCard (for Apple Macintosh) that support animation (Vertelney 1989). Such tools can animate pre-specified sequences of computer paintings and scanned pictures. For instance, the scanning allows fast and easy illustration of how a paper sketch of a screen will appear in a sequence of screens. A third possibility is provided in experimental development environments such as OSU (Lewis et al. 1989, Yang et al. 1989). Here various sorts of Apple Macintosh interface objects can be designed by direct manipulation. A proposed sequence for the appearance of application objects can be designed with a so-called Graphical Sequencer. The objects are inserted in a graph specifying the sequence of use together with triggers to step through the sequence. From the sequence graph a Pascal shell program can be generated, and the whole sketch can be used as input for the development of a final application in a CASE-tool like environment.

Finally (interactive) video can be used to illustrate and experiment with behavioural aspects of paper and cardboard mock ups as described in (Vertelney 1989). Vertelney describes the use of video recordings of step-wise changes in mock ups to animate a proposed computer use scenario. The recordings can be played as a movie to illustrate the interaction. But in more advanced set-ups, recordings can be played under computer control, making the animation resemble an interaction with a computer, the difference being that the pre-specified operations are shown on a video screen or a separate video window on the workstation.

The tools described above support the iterative development of an animation of the visual parts of the user interface. The tools are highly interactive and based on direct manipulation, hence they could support some of the idea generation and exploration aspects of cooperative prototyping. Little effort is required to develop and manipulate the screen-objects and the sequences in the design proposals. The need for paying attention to technical matters is minimized during the design process. But evaluation in work-like situations is poorly supported by the limited kind of functionality provided by these animation tools. For instance, sample data usually cannot be handled with animation tools.

4.2 Simulation

A number of tools support both the direct manipulation design of user interfaces and the simulation of functionality. Simulation in this context means the ability to provide quick-and-dirty program functionality behind the visual and tangible parts of a user interface. A program simulation can be used to illustrate how machines, such as photo copiers or printers, work. Such a simulation can then be used to design the user interface for controlling the machine. Program simulations can also be used to develop quick-and-dirty functionality behind Information System prototypes; for instance, access to remote databases can be simulated with a small local database, and data entry can also be accepted and stored without detailed validation.

An example of the simulation of a machine is provided by Trillium (Henderson 1986, Blomberg 1986), a substrate built in Interlisp-D to support design of photo copier and printer user interfaces. With this substrate it is possible to build user interfaces consisting of primitives such as buttons, controls, and displays through direct manipulation. Trillium also supports the creation and reuse of objects that are aggregations of the primitives. Behavioural dependencies between the objects on a corresponding physical machine are automatically simulated with an underlying program to let users/designers envision that they are working with a real photo copier. Trillium is a domain specific tool, supporting only the design of interfaces to ``simple'' machines that have interfaces similar to photo copiers. As such, Trillium has been successfully used to prototype user interfaces by several groups of designers working on photo copiers and printers within Xerox Corporation, due to its ability to allow experiments with the functionality underlying an interface without building the much more expensive physical machine. Trillium supports idea generation/exploration and to some extent the work-like evaluation of behavioural aspects of photo copier interfaces.

Some 4th Generation Systems for IS development, such as MANTIS[5], support the simulation of functionality behind the visual interface. In these cases a screen editor allows the designers to connect the fields from screen layouts to a data structure with search and sorting facilities that make it possible to store and retrieve a small number of test data items without programming functionality that is based on the larger and less flexible database system. These facilities make it possible to support work-like evaluation to some extent, using modest sample data, early in a design process. In other 4th Generation Systems partial functionality can be provided utilizing the integration between a screen editor and a database system. In a system such as DATAFLEX[6], database schemas can be generated automatically from screen layout specifications, or vice versa. In these cases the boundaries between a simulation and an implementation are blurred. In the IS domain such simulations would support both idea generation/exploration and work-like evaluation of ideas.

HyperCard for the Macintosh is an example of a combination of a database and hypermedia tool that allows design of graphical user interfaces and hypermedia applications exploiting fields, graphics, buttons, menus, and simple dialogue boxes. The basic metaphor is cards-in-boxes. Several functions that manipulate cards and data contents entered on cards are provided. HyperCard is not suitable for building large scale Information Systems, but it provides facilities to simulate parts of such systems within certain application domains. An example would be medical case records, where the existing manual records are a set of sheets or cards contained in folders and boxes. Bødker & Grønbæk (1989) describe an example of the use of HyperCard for cooperatively prototyping a case record system for dental clinics. They achieved a design process that closely coupled the fluent work-like evaluation of a prototype and the on-line modification of the prototype. This example was also briefly described in section 3 of this paper.

4.3 Wizard of Oz

In (Kelley 1983, Wilson & Rosenberg 1988) a so-called Wizard of Oz[7] technique is described. It is based on the idea of having a person ``sitting behind a curtain" simulating functionality of imagined programs. These techniques have been applied in domains such as design of speech recognition and command language user interfaces. In these domains the technique only requires a modest set of tools. A common example of the use of the technique is in designing command languages or speech recognition systems based on novice users' intuitive way of formulating commands for certain tasks. The experiments are carried out in sessions where users perform tasks either by talking into a microphone or by typing commands at a terminal. Depending on the precise setting of the experiment, the user input is interpreted by a hidden designer or operator who then provides reasonable and immediate feedback that appears to come from the computer. Using this technique, users and designers can get an idea of which commands and functions would be natural to have for the particular use domain. Such experiments on the development of a command language interface to an electronic mail system is described in (Good et al. 1984). They concluded, among other things, that adding numerous synonym commands to the mail system improved both expert and novice satisfaction with the system.

Using this technique with a designer/operator acting as imagined computer functionality could be a way to establish a cooperative prototyping process on certain aspects of a new computer system.

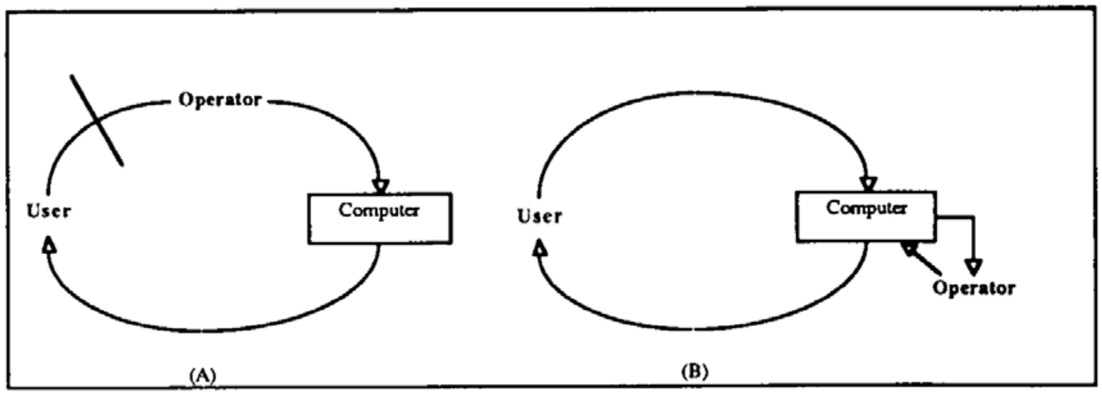


Figure 2: Two typical set-ups for Wizard of Oz experiments. In case (A) all user input is interpreted by an operator before the computer performs an appropriate operation and responds to the user. In case (B) the operator only helps interpreting commands that cannot be interpreted immediately by the computer.

But with respect to the concept of user involvement in system design presented in this paper, a serious problem is raised by the traditional use of Wizard of Oz techniques. These techniques have most commonly been applied to deceive users or test subjects participating in controlled experiments. For instance, the users participating in the experiments described in (Good et al. 1984, Kelley 1983) were not told that they were working with a system where a human being simulated large parts of the functionality. In general such approaches raise some ethical questions on how to treat users in evaluation experiments. In particular, for cooperative prototyping it is important that the activities are based on mutual trust and that the users' understanding of the situation is not distorted by deception. However, the idea behind the Wizard of Oz technique is quite exciting when considering support for easy modification of functionality and setting up work-like evaluations of prototypes. In theory the technique can be used for cooperative prototyping if the participants agree to play the game, but this still needs to be tried out. The human simulated functionality although not directly manipulable in the sense it was mentioned in section 3.2, is quite easily modified. The participants would understand the set-up with a human operator if it was explained to them (Figure 2 shows two typical set-ups). Evaluations in work-like situations are clearly possible to establish, as in the case, mentioned above, of performing electronic mail tasks.

However, it seems hard to extend the technique to simulate functionality behind full-screen and graphical user interfaces with a designer/operator sitting at another workstation. Taking a direct transfer of the technique to graphical interfaces will probably not work; for instance, it will be hard for the operator to interpret what a user expects to happen when an icon is created and then clicked. This would require the operator to ``read the user's mind" during the sessions. Thus, the technique need to be applied in different ways for this purpose. One possibility would be to have the user verbally expressing his/her expectations ("thinking aloud") on the functionality for the operator in each step of the interaction. The overhead of verbal expression, however, makes the imagined use situation less realistic than in the command language design example. Another possibility would be to have the user and the operator start off with a discussion of the detailed functionality behind certain buttons and menu items. Then the operator knows what to do when receiving a signal on his/her terminal indicating which operation is requested by the user. A third possibility would be to construct a limited set of representative tasks in advance to be performed in the session. This would also help the operator to understand what the user expects when requesting a certain operation. Finally, the imagined system response in a graphical interface will also be harder to simulate for the operator ``sitting behind the curtain"; for instance, windows, icons, etc. often need to be moved in accordance with the user's handling of the mouse. The operator need to have a ``hand" working on the user's screen. More work on such ideas needs to be done to explore whether, or how the techniques can be applied to cooperative prototyping of systems with interactive full-screen and graphical user interfaces.

4.4 Design-by-example

A category of tools to support rapid development of user interfaces and some kinds of applications is often denoted Design-by-Example or Programming-by-Example. Such tools support designers and to some extent users in (re-)designing user interfaces or final applications in certain domains through step-wise demonstration of how the user interface or application should behave. These tools are closely related to the visual programming tools discussed briefly in subsection 4.5.

Domain specific design-by-example tools for IS development are not new. Tools aimed at creating reports from existing databases through specification of a sample report in a template or a form have been marketed in several years. An example of such a tool is Query By Example (QBE), see e.g. (Gray 1984, pp. 145-154). QBE is a query facility that supports information retrieval, such as report generation from databases, by means of template filling. To request a report, a template showing the attributes of the requested table is brought up on the screen. The user of QBE then has to describe an example of the criteria that each attribute should fulfil. Afterwards the report output is automatically generated with a predetermined layout. To retrieve from single tables the user needs some familiarity with the inherent structure of the database and mathematical symbols such as ``='', ``<'',``>'', ``not'', ``and'', and ``or''. To make combinations from multiple tables more advanced command syntax describing attribute selection is required. These tools can support cooperative prototyping when designing reports for Information Systems. Laying-out reports is a common task and traditional programming can be kept to a minimum in prototyping sessions if the standardized lay-out styles are satisfactory. Assuming that a sample database can be provided, these tools also allow work-like evaluation of prototypes.

An example of a more general design-by-example tool is a User Interface Management System for graphical user interfaces called Peridot (Myers 1987), implemented in Interlisp-D. Peridot lets the designer design user interface dialogues by drawing screen objects in a display window after which they are immediately usable for interaction. Peridot in this way allows the designer to ``program'' the dynamics of the user interface by demonstrating the user's mouse movements and clicking on menus, buttons, scroll bars, and windows etc. A virtual 3-button mouse appearing as an icon on the screen is the representation of the real mouse. For instance, moving this icon on top of an already drawn menu simulates the use of the menu. This way it is possible to specify whether an item is chosen by pressing a button or by releasing a mouse button while over an item. Toggling, checking, and inversion of a menu item can also be demonstrated. The interfaces created with Peridot can be tried out immediately as stand-alones or in conjunction with the application program that the user interface is being designed for. Moreover, conventions that can be deduced from previous demonstrations are recorded automatically in a rule base and then re-applied when new objects similar to existing objects are created.

Myers claims that ``Peridot can create almost all of the Apple Macintosh interface, as well as many new interfaces such as those using multiple input devices concurrently'' and ``Nonprogrammers can use it to create user interfaces'' (Myers 1987, p. 59). If true, it means that Peridot is a powerful tool for designing graphical user interfaces without traditional programming activities. Thus, Peridot seems to be a potential source of ideas for developing tools for cooperative prototyping. A problem to be solved, however, is that the user interface design is restricted technically to using menus, windows, etc. as the primitives. In a situation where users and designers work together on designing an application for a specific domain it is important to be able to aggregate and specialize the primitives. Such facilities are useful in developing objects that correspond more directly to objects well-known from the use domain, to help users understand what is going on. To turn a powerful tool such as Peridot into a tool for cooperative prototyping for a certain application domain, a ``palette'' of domain specific objects needs to be accessible.

4.5 Visual programming

The final group of tools to be discussed here is denoted visual programming systems. The term visual programming indicates that such systems attempt to substitute point-and-select programming for traditional programming. For example diagram manipulation, template-filling, menu-selection, and clicking of icons is used instead of writing textual programs. This is not totally distinct from some of the tools described above. In fact a number these systems, e.g. OSU, QBE, and Peridot, have occasionally been labelled visual programming systems because of their substitution of direct manipulation programming for traditional programming.

Recently, however, attempts have been made to develop fully visual and general programming systems to substitute for traditional programming languages. An example of such a system is an object-oriented programming environment called Prograph[8]. This environment provides: direct manipulation design of screen objects, fully iconic/graph based program editing, graphical browsing to edit specialization hierarchies, interpretation based execution, and debugging based on animation of the diagram/icon program, visual showing of the run-time stack, and finally support for persistent objects. See Figure 1 for an example of how parts of the programming environment appear on the screen. These features makes Prograph quite fascinating to programmers/designers at a first glance. But beyond user interface design, programming in the environment still involves a lot of low level programming details, although it is done by drawing lines and placing icons. This fact makes it still unlikely users could be directly involved in programming system functionality with Prograph. But the highly interactive graphical browsers and editors together with interpretation based execution, could potentially support idea exploration aspects of cooperative prototyping. Moreover, persistent objects support convenient handling of the sample data needed for work-like evaluations of prototypes. But as with the design-by-example tools, it is needed to build a library of domain specific objects embedded in the general visual programming system for easy reuse/specialization in order to involve users in modifying prototypes.

4.6 Overloading direct manipulation?

In this section a number of direct manipulation design tools and their potential for supporting cooperative prototyping have been discussed. Direct manipulation support for this purpose seem to be quite attractive and promising. But it should not be ignored that direct manipulation tools also have some drawbacks. Shneiderman (1983) coined the term ``direct manipulation" and named it as one of the most promising approaches to the development of human-computer interfaces. Since then some authors have been critical of the idea of transforming all sorts of human-computer interaction into direct manipulation. In (Hutchins et al. 1986) it is argued that direct manipulation tools appear to be tedious and difficult to use in a number of cases such as: performing repetitive operations, representing variables, and providing flexibility and efficiency to expert users. It is also claimed that direct manipulation tools narrow the borders of innovation. Their concluding claim is that general and powerful programming languages never will or should be replaced with direct manipulation tools.

These critical remarks surely hold for some direct manipulation facilities; for instance, it becomes tedious for an experienced programmer to pick up icons for each program statement as is the case with Prograph described above. Program statements can typically be typed in faster from the keyboard once you are familiar with the syntax. Hence it is not a good idea to handle all human-computer interaction by direct manipulation, even though it might be technically possible. But, as argued earlier, users and designers who want to perform cooperative prototyping need concrete and tangible models to refer to. Thus, they need direct manipulation support for much more of the programming activities than do the programmers/designers who are programming on their own.

To summarize, the development of direct manipulation tools embedded in powerful object-oriented programming environments seem to be a good direction to move when developing computer support for cooperative prototyping. Direct manipulation tools can be used in particular to stimulate user creativity in prototyping sessions. The more powerful underlying programming environment can support the designers' creativity in designing and extending the set of domain specific objects to be used in a direct manipulation fashion in cooperative aspects of the design and evaluation process.

5. Concluding remarks

It has been argued that the prototyping approaches currently applied in practice are too limited with respect to support user involvement and creativity. The approaches typically adopt only the project manager's or designer's perspective. This limitation is a crucial obstacle to development of high quality computer systems tailored to the users' needs. A cooperative approach to prototyping is proposed, to stimulate user and designer creativity in particular early in a development process. Successful prototyping requires that users and designers are able to work with tangible models of future systems. This in turn requires better prototyping support. Hence, a brief survey of potential tools to support cooperative prototyping in various use domains is given. The tools are in general, however, not ready to be taken from the shelf for use in cooperative prototyping. They do, however, point to directions for further development of computer support for cooperative prototyping, both in general and in specific development organizations.

Having stated that cooperative prototyping is a technique to be used particularly in early development activities, a few remarks on development continuation are needed. The tangible models described often cannot be reused in final applications, but this does not imply that users should only be involved in the early stages of system development. However, goals of development should be clarified and alternatives explored through cooperative prototyping before the more targeted approaches are used to produce a system. Hence, to produce a system the cooperative prototyping techniques should turn into an incremental prototyping approach with emphasis on work-like evaluations of meaningful parts of the prototype that incrementally become the system. To propagate the ideas learned in early cooperative prototyping it is crucial to have some of the same people continue with the more detailed design and implementation of the new system. Lessons learned from cooperative prototyping are hard to propagate by handing over a prototype and a description. It is also important to maintain a continuous, but perhaps less frequent, user involvement throughout the full project, in order to accommodate organizational changes etc. that occur while the project is running. To summarize, the aim of the cooperative prototyping approach is to utilize more concrete media, tangible models, for the early design activities in order to allow *both* users *and* designers to be creative at this stage. Later on in the development users will act more as evaluators participating in a step-wise implementation and work-like test of the system that have been designed cooperatively.

Notes

* Full affiliation: University of Aarhus, Computer Science Dept., Ny Munkegade 116, DK 8000 århus C, Denmark. Phone: +45 86 12 71 88 ext. 5057. E- mail: kgronbak@daimi.aau.dk.

1. Refer for instance to DATAMATION July 1, 1988 for a number of articles on CASE-tools

2. Refer to (Bødker 1987a) for a discussion of transparency of tools.

3. Substrate means an ``Underlying layer'', and in this context it denotes a collection of system objects specialized for a certain application domain.

4. ORACLE and SQL-FORMS are trade marks of ORACLE Corporation.

5. MANTIS is a trade mark for Cincom Systems, Inc.

6. DATAFLEX is a trade mark of Data Access Corporation, Florida, USA.

7. The name ``Wizard of Oz'' is taken from a novel and movie in which a small man hidden behind a curtain manipulates auditory and visual effects to convince people that he is a powerful wizard.

8. Prograph is a trade mark of Sun Gunakara Systems ltd., Halifax, Canada.

Acknowledgements

I would like to thank Susanne Bødker, Jonathan Grudin, Morten Kyng, Randall Trigg, the members of working group 7 at the 12'th IRIS conference held in Skagen, Denmark, August 1989, and the anonymous referees for their valuable comments on draft versions of this paper. Moreover, this work has been supported by *The Danish Natural Science Research Council*, FTU grant no. 5.17.5.1.30.

References

Peter Bøgh Andersen, editor, (1987). Research Programme on Computer Support in Cooperative Design and Communication. *IR 70, Computer Science Department*, Aarhus University, Århus.

Jørgen Bansler, (1989). Systems Development Research in Scandinavia: Three Theoretical Schools. *Scandinavian Journal of Information Systems*, 1(1):3--20.

Jeanette Blomberg, (1986). The variable impact of computer technologies in the organization of work activities. In Don Petersen, editor, *Proceedings from the Conference on Computer Supported Cooperative Work*, Austin, Texas, pages 35--42. MCC Software Technology Program.

Bernhard H. Boar, (1984). *Application Prototyping - A requirements definition strategy for the 80s*. John Wiley and Sons, Inc., New York.

Barry W. Boehm, (1981). *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, New Jersey.

Barry W. Boehm, Terence E. Gray, and Thomas Seewaldt, (1984). Prototyping versus specifying: A multiproject experiment. *IEEE Transactions on Software Engineering,* 10(3):290--303.

Susanne Bødker, (1987). Prototyping revisited - design with users in a cooperative setting. DAIMI PB 233, Computer Science Dept, Aarhus University, Ny Munkegade 116, DK 8000 Aarhus C - Denmark, September 1987. *In proceedings of the 10th IRIS* seminar held in Vaskivesi, Finland.

Susanne Bødker, (1987). Through the interface --- a human activity approach to user interface design. *PB 224, Computer Science Department,* Aarhus University, århus.

Susanne Bødker, Pelle Ehn, John Kammersgaard, Morten Kyng, and Yngve Sundblad, (1987). A utopian experience: On design of powerful computer-based tools for skilled graphic workers. In Gro Bjerknes, Pelle Ehn, and Morten Kyng, editors, *Computers and Democracy - A Scandinavian Challenge*, pages 251--278. Avebury, Aldershot, England.

Susanne Bødker, Pelle Ehn, Jørgen L. Knudsen, Morten Kyng, and Kim H. Madsen, (1988). Computer support for cooperative design. In Deborah Tatar, editor, *Proceedings of Conference on CSCW,* Portland, Oregon, September 1988, pages 377--394, ACM, New York.

Susanne Bødker and Kaj Grønbæk, (1989). Cooperative prototyping experiments - users and designers envision a dental case record system. In John Bowers and Steve Benford, editors, *Proceedings of the first EC-CSCW '89*, Gatwick, UK, pages 243--257. Computer Sciences Company.

Susanne Bødker and Kaj Grønbæk, (1990). Cooperative prototyping - users and designers in mutual activity. Draft submitted for publication.

Richard G. Canning, (1981). Developing systems by prototyping. *Edp Analyzer*, 19(9):1--14.

Andrew Friedman and Dominic Cornford, (1987). Strategies for meeting user demands. In Gro Bjerknes, Pelle Ehn, and Morten Kyng, editors, *Computers and Democracy*. AVEBURY, Aldershot, pages 137--162.

Michael D. Good, John A. Whiteside, Dennis R. Wixon, and Sandra J. Jones, (1984). Building a user-derived interface. *Communications of the ACM,* 27(10):1032--1043.

Peter Gray, (1984). *Logic, algebra and databases*. Ellis Horwood ltd., Chichester.

Kaj Grønbæk, (1989). Rapid prototyping with fourth generation systems - an empirical study. *OFFICE: Technology and People*, 5(2):105--125. Also available as DAIMI PB 270, Aarhus University.

Jan Hauerslev and Niels Jakobsen, (1988). SOKRATES: Sprogspil og kvalifikationer - en rapport om arbejdet med en teori for erkendelsesprocesserne i systemudvikling. *Master thesis - Unpublished report made at the Computer Science Dept*., Aarhus University.

Sharam Hekmatpour and Darrel Ince, (1988). *Software Prototyping, Formal Methods and VDM*. Addison-Wesley, Wokingham, England.

D. Austin Henderson, (1986). The Trillium User Interface Design Environment. In Marilyn Mantei and Peter Orbeton, editors, *CHI '86 Proceedings*. ACM, pages 221--227.

Pei Hsia and Alan T. Yaung, (1988). Screen-based scenario generator: A tool for scenario-based prototyping. In Bruce D. Schriver, editor, *Proceedings of the Twenty-First Annual Hawaii International Conference on System Sciences*, Vol. II, Massachusetts Av., Washington D.C., pages 455--461. Computer Society Press of the IEEE.

E. L. Hutchins, J. D. Hollan, and D. A. Norman, (1986). Direct manipulation interfaces. In Donald A. Norman and Stephen W. Draper, editors, *User Centered System Design - New Perspectives on Human-Computer Interaction*, chapter 5. Lawrence Erlbaum Associates, New Jersey.

J. F. Kelley, (1983). An empirical methodology for writing user-friendly natural language computer applications. In Ann Janda, editor, *CHI '83 Proceedings*, Human Factors in Computing Systems, Boston, Mass., December 12-15., pages 193--196, ACM, New York.

Kenneth E. Lantz, (1986). *The Prototyping Methodology*. Prentice Hall, Englewood Cliffs.

T. G. Lewis, Fred Handloser, Sharade Bose, and Sherry Yang, (1989). Prototypes From Standard User Interface Management Systems. In Bruce D. Shriver, editor, *Proceedings of the 22nd annual Hawaii International Conference on System Sciences*, Vol II, volume 2, pages 397--406, Washington D.C. Computer Society of the IEEE.

Klaus Viby Mogensen, (1985). Afprøvning af systemudvikling med prototyper. *DIKU-report 85 10*, Institute of Computer Science, University of Copenhagen, Copenhagen, Denmark. Masterthesis.

Brad A. Myers, (1987). Creating Interaction Techniques by Demonstration. *IEEE Computer Graphics and Applications*, 7(9): 51--60.

Ben Shneiderman, (1983). Direct Manipulation: A Step Beyond Programming Languages. *Computer*, 16(8): 57--72.

Stephen L. Squires, Martha Branstad, and Marvin Zelkowitz, editors, (1982). Special issue on rapid prototyping. *Software Engineering Notes*, Vol 7. 5, ACM SIGSOFT, Baltimore. Working papers from ACM SIGSOFT Rapid Prototyping Workshop, April 1982.

Steen Thomsen, (1987). Experiences with system specification by prototyping. *Presented at the EFISS-87 conference*, Roskilde, Denmark.

Laurie Vertelney, (1989). Using video to prototype user interfaces. *SIGCHI Bulletin*, 21(2):57--61.

James Wilson and Daniel Rosenberg, (1988). Rapid prototyping for user interface design. In Martin Helander, editor, *Handbook of Human-Computer Interaction*. North-Holland, Amsterdam, pages 859--876.

Terry Winograd and Fernando Flores, (1986). *Understanding Computers and Cognition*. Ablex Publishing Corp., Norwood, New Jersey.

Sherry Yang, T. G. Lewis, and C. C. Hsieh, (1989). Integrating computer-aided software engineering and user interface management systems. In Bruce D. Shriver, editor, *Proceedings of the 22nd annual Hawaii International Conference on System Sciences*, Vol II, volume 2, pages 850--859, Washington D.C., Computer Society of the IEEE.