STEPS to Software Development with Users

Christiane Floyd, Fanny-Michaela Reisin, Gerhard Schmidt
Technical University of Berlin
Department of Computer Science
Franklinstraße 28/29
D-1000 Berlin 10

Abstract

The paper reports on the methodical approach STEPS, developed at the Technical University of Berlin and tried successfully in participative software development with users. STEPS views software development in its connection to work design. It gives guidance to developers and users for carrying out their cooperation, establishing quality criteria pertaining to software in use and putting them into practice in system design. It embodies an evolutionary approach, portraying system development in cycles of version production, application and revision. It supports mutual learning by developers and users by carefully establishing and coordinating processes of cooperation, by using prototyping for experiments and by adapting methods and tools to the needs of cooperation and incremental work.

1. Introduction

I would like to thank the organizers for having invited me (Christiane Floyd) to address the topic of user-oriented software development at this conference. In doing so, they acknowledge this issue as an important and legitimate concern for the software engineering community. As I have worked on re-orienting software engineering methodology to support software development with users for several years with my colleagues at the Technical University of Berlin, I have chosen to write as a response to the invitation a joint paper with my co-workers Michaela Reisin and Gerhard Schmidt who have carried out with me the conceptual and empirical work reported here.

To recognize the importance of this line of work from the point of view of software engineering is not at all obvious : evidently, one might argue, all software is intended for use, therefore all software development is user-orieneted. Why, then, should user-oriented software development be a topic requiring

special treatment? We maintain that sound approaches to software development as provided by software engineering methodology will lead to high quality products, which has been the concern of the field all along, and it seems legitimate to argue that this is as far as our responsibility goes.

I see this position implied by the mainstream of software engineering work. I believe, that it is held by most of my colleagues and students, and I know, it does not necessarily imply a lack of consideration on their part for user concerns. Rather, it relates to basic assumptions on the nature and scope of scientific work in our field, concerning in particular the borderline between software engineering and other disciplines on one hand and the separability of production from the use of products in social contexts on the other hand. This position is in keeping with that held predominantly in other engineering disciplines and implemented by society at large.

As a result, I perceive a gap between efforts of two kinds : "technical" efforts from within software-engineering aimed at providing high-quality products and "social" efforts aimed at promoting conditions for the use of these products oriented to human values. The latter are considered to be separable from and outside the scope of software engineering.

By contrast, my starting point for the present discussion is to question the actual separability of these concerns. On the basis of practical work in projects as well as in developing and evaluationg methods, I have been led to believe that viewing software development as production, and thereby focussing our attention primarily on the product software, is misleading. Instead, I consider processes of software development as the primary area of concern, I regard the product software as emerging from these processes and the use of software intertwined with its development. As a result, I have come to view software development as design rather than as production. This work has been reported in a series of papers (/Floyd 81/, /Floyd, Keil 83/, /Floyd 85/, /Floyd 86/, /Floyd 87/, /Floyd 89/).

Against this background, we are in the process of developing and evaluating at the Technical University of Berlin a methodical approach STEPS (Software Technology for Evolutionary Participative System Development) supporting, in particular, the cooperative development of software with users. The development effort for STEPS and its evaluation in academic and industrial projects has been going on for several years. Besides the authors of the present paper, there are other major contributors (see for example /Keil-Slawik 89/ on task-oriented requirements analysis and /Pasch 89/ on self-organization in software development projects).

Recently, STEPS has been tried successfully in the PEtS-project in a real-life participative development of an information system for handling archives of union contracts in Germany (see /Floyd, Mehl, Reisin, Schmidt, Wolf 89a/). This project was funded as part of the research programme "Man and Technology: Socially-Oriented Technology Design" in North Rhine-Westphalia.

While it is difficult to generalize the results of one project to other settings, we have nevertheless become confident that in information technology there is a potential for reconciling the concerns of rationalization and humanistic work design that has not yet been tapped in larges scale system development. We hope to make a contribution in this direction through our work and to encourage others to proceed along this line.

In this paper, we concentrate on the basic assumptions and methodological questions associated with STEPS.

Section 2 outlines of STEPS as an overall approach. A key issue is its perspective on software development, considering the anticipation of use an inherent part of the development process and the tasks of software developers implicitly related to work design. While this perspective, in principle, is relevant for all software, it becomes one of predominant concern for interactive application systems, that is the class of all software systems to be fitted into work processes. In this context, software development does not start from pre-defined problems, but must be considered a learning process involving the unfoldment of the problem as well as the elaboration of a solution fitiing the problem.

In section 3 we point to theories for understanding design as a process. In particular, we examine the notions of evolution and participation as employed in STEPS. Evolution, here, refers to the emergence of insights into the functionality and the potential use of computer programs. Software development is a cooperative cognitive process akin, in particular, to design in other fields. Participation is a term used mainly for historical reasons. As seen from now, it refers to a class of actual strategies for cooperation between developers and users, where we adopt that of mutual learning.

In section 4 we present the project model used as the basis for our work. It portrays software development as a sequence of cycles leading to the production and application of a system version. A system version consists of technical parts in the form of executable computer programs and their defining documents, supplemented by guidelines for the organization of work to be supported by the programs. A situation-specific development strategy consists in choosing the scope and the number of versions, the frequency of development cycles and the modalities for actual cooperation between developers and users.

In section 5 explain the dynamic coordination of the development process through project (and cycle) establishment and reference lines based on our experiences in actual projects The idea, here, is to synchronize the taking and meeting of responsibilities by all participants in view of obtaining and evaluating partial results in a step by step manner.

Section 6 clarifies the role of methods and tools in cooperative software development : the use of prototyping to support mutual learning, the adaptation of methods to communicative settings and the potential of tools for supporting cooperation and incremental work.

Finally, in section 7 we place our work in its social context. An approach like STEPS may enable us to carry out software development work systematically in an open social setting marked by interests, conflicts and change. However, no method can be expected to replace political measures allowing participation. Nor can it serve as a substitute for human attitudes required for cooperative work. We conclude by stating our conviction that facilitating processes of software development with users ultimately rests on our human abilities for dialogue and mutual acceptance.

2. Outline of STEPS

STEPS is concerned with the cooperative development of software to be fitted into user work processes. It views software development as design, and considers the anticipation of use an inherent part of the development. As a consequence of this orientation, it has strong connections with other efforts both in software engineering and in participative system development.

As methodical approach to software development, STEPS is closely related to the view of programming as "theory building" presented in /Naur 85/. It is in keeping with the insight put forth by /Parnas, Clements 85/ on how "rational" software development processes are "faked" counterparts of what actually takes place. It draws consciously on the idea of software evolution expressed so powerfully in /Lehmann 80/. Lastly, there are important connections to ongoing research efforts in the field of prototyping (see, for example, /Budde, Kuhlenkamp, Mathiassen, Züllighoven 85/).

As approach to participative system development, STEPS is related to pioneering efforts in England (/Mumford 87/) and in Scandinavia. In order to be able to profit from these ideas and experiences in our own work, we have compiled a study on methodological concepts and innovative projects for user-oriented information system design in Scandinavia (/Floyd, Mehl, Reisin, Schmidt, Wolf 89b/). Our own work has been strongly influenced by our long term contacts with authors from the so-called Scandinavian school centered in Oslo and in Aarhus (see, for example, /Nygaard 87/, /Andersen et al. 86/, /UTOPIA 85/,/Ehn 88/,/Bjerknes, Bratteteig 87b/).

In the application area addressed by STEPS, software developers provide tools to be used by others and media for the communication between others. That is, our technical concerns for providing high-quality products are inherently tied up with issues of communication, work, and social processes, which define the very nature of the problems that we deal with.

While software developers are directly concerned with programming, they contribute indirectly to profound changes in the working life of the users of their programs. Therefore, software developers have

to understand user work processes, the potential role of the computer as an artifact, and the scope for design available taking account of the actual situation and the interests of all people involved.

Users, on the other hand, are faced with a revision of their work processes. This involves re-organization of work, the acquisition of new skills in using computer programs, and far-reaching changes in competence. STEPS aims at supporting software development so as to enhance user competence. This implies both acquiring new competence (in handling the computer) and enriching existing competence by gaining a renewed understanding of the work in hand and of the potential of the computer to support it.

In keeping with this, we consider software development a learning process for both developers and users.

STEPS, thus, is an approach to software development taking account of its connections to work design. The content, organization and conditions of work determine the quality and structure of the software system to be developed. Conversely, the software system has an impact on the working processes. Crucial for judging the quality of a computer-based system from the users point of view, is an evaluation of the quality of their work when using the software. This gives rise to our view of system development, which is not restricted to the product software, but also takes into account the social processes and relations in the context of which it is produced and used.

Those participating in a software development project are creative in two respects: In designing a computer-based system, they are creating and shaping a product, and, at the same time, the development process itself. These two dimensions are of a reciprocal nature. They are usually, though implicitly, reflected in the course of system development by two classes of different activities: product-oriented and process-oriented activities.

STEPS provides guidance to developers and users for carrying out their cooperation in actual development projects concerning both product-oriented and process-oriented activities. It does not claim, however, to furnish generally applicable criteria on how computer-supported work should be designed. To the extent that such criteria are available at this time, we consider them to be within the realm of social theory. Rather than supplying ad-hoc criteria of our own, we aim at showing ways for how such criteria can be accommodated and put into practice in software development.

STEPS relies on perspectivity in a number of ways. By "perspective" we mean a class of related views on relevant aspects of an area of concern from a common view-point. Perspectives arise both from our individual subjectivity and from positions adopted by goups and related to common tasks or formulated interests. While perspectives are always implied in our thinking, we can attempt to make them explicit, allow them to interact and gain deeper insights from their interrelation.

In software development, this means above all to recognize the "use perpective" held by those who interact with software as a tool or as a medium as distinct from the "development perspective" held by software developers Furthermore, there is no one single use perspective, but a plurality of perspectives related to functional roles of different users, to collective interests and to individual tastes and priorities. Different perspectives on one and the same software system may be in conflict due to misunderstandings or clashes of interest. If we take working with users seriously, these conflicts must be acknowledged and dealt with rather than smoothed out in a quasi-objective system analysis.

Thus we see in multiperspectivity a basic prerequisite for cooperative work.

This leads to concrete consequences for software development. While traditional approaches tend to advocate the early construction of comprehensive and quasi-objective models relying largely on formal techniques to be used in defining documents, the emphasis here is on recurring to perspective-based evaluation as the supreme guide both in building models and in interpretating constructed models in the context of meaningful human activities.

Therefore, we relate software requirements to the context of user work processes as a whole (see /Keil-Slawik 89/), rather than restricting our attention to desired functions of the software system as advocated by other existing software development methods (for example /Jackson 83/, see also my evaluation in /Floyd 87/).

Treated in this manner, requirements and system functions become distinct domains of discourse, which are relevant throughout system development and are connected actively through design. Requirements are not "given" and therefore cannot, strictly speaking, be analyzed. Their gradual establishment takes place in an interplay of anticipative, constructive and evaluative steps to be carried out by developers and users in interaction.

Owing to ongoing social changes, it is not possible to define the required functions and quality of a software system, which is to be used in working processes, completley at any fixed point in time. On the one hand, requirements evolve because of general changes in society, the economy, technology, law etc.. On the other hand, changes in work organization, users qualifications etc., which are not least an effect of the software system itself, give rise to new and changing requirements. In order to keep pace with changing social and economic interests, software development must be based on the evolutionary development of the use organization and, in particular, of the work context in which the software is directly applied.

On the technical side, therefore, STEPS embodies an evolutionary approach comprising various forms of prototyping and the development of systems in versions. This approach can be visualized in a static project model that allows to choose from a class of situation-specific strategies as needed for the project in hand.

While a project model is useful as a map showing salient features of the territory of interest, the actual cooperative process must be initiated and coordinated dynamically in the course of the project itself.

The development, then, consists of an interleavement of some activities to be carried out jointly by developers and users and of others to be carried out by the respective groups on their own. Conventional software development methods for requirements analysis and design have their place, but need to be tailored to the needs of the communicative processes at hand, so as to show multiperspectivity and to allow incremental work.

STEPS embodies a human-centered notion of quality emphasizing the experience with software in use as primary level of concern. We consider it important to give precedence to the specific user-oriented quality criteria, which are derived from the application area and related to the people working there - such as handleability, amenity, intelligibility etc. - over the general technical quality criteria, such as reliability, portability, compatibility which we consider to be necessary prerequisites for meeting user-quality criteria.

In order to make up the specific product quality, it is important to focus not only on the formal aspects of the application area. Informal work factors, differing perspectives, conflicting interests and divergent preferences must be given attention as well. Therefore we argue that user-oriented quality criteria can only be established on the basis of a communicative process of mutual learning (/Bødker 86; Bjerknes, Bratteteig 87a/).

STEPS aims at facilitating the emergence of quality experienced by all participants through cooperative design.

3. Understanding Design as a Process

Software development has been characterized in /Naur 74/ as an "activity of overall design with an experimental attitude". This holistic view is very much in keeping with STEPS. We hold that design is not restricted to specific development stages or to selected specified areas of concern but is pervasive.in all development activities. Thus, while there is a development step "system design" in the project model explained in section 4, we do not mean to suggest that design is confined to this step only.

In software development, design pertains to the establishment of requirements and their connection to software functions and archtiecture, to the realization of programs, to the anticipation of software use and the adaptation of software for creating computer-based milieus in different settings, and of course, to the

development process itself. This list is not meant to be exhaustive. On the contrary, there can be no exhaustive enumeration of design concerns, as they arise and are dealt with as needed in the ongoing process.

Design is not primarily tied up with the achievement of pre-defined goals, but is guided by insights emerging in the actual process and by the quest for quality shared by all participants. While, of course, design is always connected with goals, we must allow for an examination and revision of these goals as part of design (for more details on this view see /Floyd 89/).

The study of design in software development is clearly outside the scope of the present paper. However, we feel the need to stress our conviction that sound methodical approaches to software development must ultimately rest on a well-founded understanding of the nature of design taking careful consideration of its epistemological, social and technical dimensions and their interrelation This concern is at present pursued by a number of researchers committed to establishing foundations for design from various view-points (see, for example, /Winograd, Flores 86/, /Ehn 88/ and /Budde, Floyd, Keil-Slawik, Züllighoven 89/). Here, we will restrict ourselves to sketching some avenues of research that seem relevant to us for understanding design as a process, and that we pursue in depth as part of the research associated with STEPS.

The view of programming as "theory building" presented in /Naur 85/ with reference to /Ryle 63/ provides an excellent frame of reference for making the connection between the growing and ever-changing understanding of the application area and the relevant design decisions embodied by programs - constituting the "theory" held by programmers - on one hand, and the programs reflecting this theory on the other hand. Programs, as well as defining documents, can only make some aspects of their underlying theory explicit. The theory itself remains inherently with the programmers. The "life" and "death" of programs can then be related to their development processes carried out by individuals or teams who possess the theory for the program. Though this is not made explicit in /Naur 85/, we may apply this concept in a similar manner to users building a theory on how to apply programs in the context of their work processes.

As a cooperative cognitive process, design can be elucidated by the ideas of cybernetic and evolutionary epistemologists. For example, we can draw useful insights from considering projects as exhibiting "mind" in the sense of /Bateson 80/; the objective of methodical approaches, then, is to promote the unfoldment of mind in software projects. Or we may study design under aspects like self-organization (/v. Foerster 60/) and autopoiesis (/Maturana, Varela, Uribe 74/) as a basis for facilitating the cognitive processes in hand. And we can use dialogical concepts (/Bråten 73/, /Bråten 78/), and learn from conversation theory (/Pask 75/) for understanding the interaction of different perspectives in design (see also /Pasch 89/).

While these theories provide fascinating accounts pertaining to the emergence of insights, they take little notice of fundamental factors such as subjectivity, motivation, individual and collective interests, power and conflict, as constitutive elements of design as social process. These merit a study of design along very different lines, as provided by activity theory (/Leontiev 78/, /Vygotsky 62/, /Holzkamp 78/, see /Engeström 87/ for an attempt at applying this theory to matters of concern in our area).

Yet another important approach is to study design as an organizational process (following, for example, /Weick 79/) in order to gain insights for how to conduct software development projects.

Design takes place as a process of cooperation amongst software developers and users. Mainly for historical reasons, we use the established term "participation" in this context.

However, we have come to consider this notion as misleading. For one thing, it does not clarify, who is participating. If everyone is participating, the term is empty. If users are supposed to participate, then the term suggests a lop-sided arrangement, where developers carry the bulk of responsibility and the ultimate power for making decisions. In STEPS, we do not have such an arrangement in mind.

The competence needed for designing technology and work, and in particular for determining user-oriented quality criteria, is not possessed unconditionally either by the users or the developers. Their joint participation in the system development process constitutes a necessary condition for creating the new knowledge required and for a shared expertise for design. If software development is to aim at improving the quality of work, the methods and organizational forms used must be tailored to support cooperation between users and developers, i.e. to facilitate communication and mutual learning processes during system design.

We aim at supporting processes of mutual learning, where developers and users have complementary competences of equal rank, and where decisions pertaining to the design of software as a tool or as a medium ultimately rest with the users.

4. A Project Model for Software Development with Users

As a basis for systematic work in software development, STEPS relies on a project model as shown in figure 1. This model serves as a map to guide all participants into the territory of software development by establishing a common understanding of the tasks to be performed. As can be seen, the project model is distinct from conventional life cycle models in several ways:

- It is cyclical rather than linear, portraying software development as proceeding in system versions, all development steps and defining documents therefore being subject to revision.

- It combines software production and application, visualizing the tasks of developers and users rather than those of developers only; the development of each technical system version is fitted to an associated re-organization of user work processes; the interplay between these two is anticipated in the ccoperative design and evaluated in the cooperative revision step.

- It refers to a class of possible development strategies allowing the choice of a situation-specific strategy as needed in the project at hand rather than depicting one ideal development strategy to be copied as closely as possible in all projects. A situation-specific strategy consists in planning the development cycles and the prospective scope of the system versions in keeping with the needs of the work processes, with available resources and actual modalities for cooperation.

- It relies on a minimum of pre-defined intermediate products, thus allowing the freedom for choosing the actual intermediate products as needed in the ongoing process; intermediate products are system versions as well as defining documents, they are subject to evaluation and planned revision.

- It is not defined in terms of the domains of discourse relevant in software development (requirements, systems functions and architecture, executable program components etc.) and thus avoids suggesting that these should be treated sequentially; rather it assumes that they are relevant throughout all development cycles. They need to be separated and re-connected in system design, and our understanding of them increases in successive cycles of production, application and subsequent revision.

Figure 1: Project Model of STEPS

- It incorporates the dynamic coordination of the ongoing project in the form of the project (and version) establishment.. In the course of the project, the dynamic coordination continues through reference lines. These, however, are not shown in the model, as they occur as needed during the actual process (see section 5).

We consider system design, software realization and use to be the most important activities in software development.

In *system design*, the quality and structure of the software version and the computer-supported work associated with it are established, evaluated, and determined by the users and developers cooperatively.

They decide and agree upon issues of different kinds such as working activities and work organization, requirements, system functions, data structures and representation, the user interface, hardware and software components to be provided or purchased, and organizational embedment of the computer-based work.

System design is carried out in cycles of analysis, synthesis and revision. Attention focusses here on the communication and learning processes occurring between users and developers, during which a new reality domain is created. In contrast to traditional views, we hold that in system design processes of communication and mutal learning are the really creative part of the work leading to insights. All other activities, such as formalization, description and implementation, are considered to be of subordinate significance.

Software realization by the developers comprises the design of the software architecture as well as the effect specification, implementation, testing etc. of the system functions. Here, too, design is carried out in cycles of analysis, synthesis and revision, these being supported by suitable design and implementation methods and techniques.

Use of the software system refers to its employment by the users in the course of their daily work.

For a particular development cycle the project model calls for two predefined products only : the system specification and the system version.

The *system specification*, the result of system design, is produced by the developers. It contains the functional specification for the system version to be realized, besides specifying the qualification and procurement measures to be taken, organizational changes, alternations in room arrangements etc. in the user organization.

The *system version* incorporates, in addition to the realized software version, the application computer and all the documents needed for the use and maintenance of the system.

In view of the wide variety of development situations, a project model reflecting generalized knowledge can provide only a framework for describing the *domains of discourse* within which software development takes place and that are anchored in different domains of the human life world. in STEPS we distinguish domains of discourse pertaining to software development as follows :

- *Requirements* are anchored in the world of user work processes.

- *Systems functions and architecture* arise as a result of design and draw on modelling concepts from the formal world of methods.

- *Programs* serve to control the computer and thus form the connection to the technical world of realization means.

Each domain of discourse is associoated with specific tasks, and its own respective methods, techniques and means of representation. The relationship between the various domains of discourse is a logical one, not equatable with the temporal sequence in which they are treated. Any domain of discourse may form the working and communication context at any point in time.

This has strong bearings on our view of design-principles such as "top-down development" and "separation of concerns". Where they are meant to be applied to the temporal organization of treatment of different domains of discourse, we find them thorougly misleading. On the other hand, they provide excellent criteria for structuring the results of design

5. Dynamic Coordination of Cooperative Work

The project as a whole, and hence each project cycle, is initiated by the users and developers by means of a cooperative *project establishment* (/Andersen et al. 86/).

Prior to the actual project start, it must be determined which tasks should be supported and which problems addressed by the system under development. In particular, those involved in the project must be prepared to engage in cooperative working processes for developing the system, and agree on a common course of action with its resultant commitments. An inevitable precondition of participative system development is that an agreement on the project's goal and on the way it is going to be achieved has to be

worked out, not only between the contractors but especially between the users and developers who are to cooperate together. In the course of project establishment, the first development cycle is initiated, too. This results is a rough *system concept,* from which the product-oriented activities are derived, and a *project strategy,* which the process-oriented activities refer to. Each subsequent project cycle is initiated by a *revision establishment* which is carried out on the basis of new or changed requirements resulting from the use of a version. As in project establishment, the users and developers agree on the new or changed requirements which have to be taken into account, and plan the execution of the revision of the version currently in use. During the revision establishment, the system concept and the project plan are updated.

To enable a more precise coordination of the development process, and at the same time to allow consideration to be given to the concrete and ever-changing project situation, we make use of the concept of *reference lines* (/Andersen et al. 86/).

Starting from the project establishment, the participants determine reference lines as needed in their working processes, defining project states to be reached in terms of intermediate products. Each reference line is named, and specifies one or more intermediate products, criteria for evaluating them, and procedures for decision-making. The next reference line has to be defined when a thus defined project state is reached, at the latest. Hence, we can prevent the project from attaining an undefined state.

Starting from the initial project establishment (PE), in the following example the first two reference lines (R1 and R2) are defined by the developers and the users. Part of reference line R2 is to define the subsequent reference line(s), here R3.

Figure 2: Coordination of a Process by Reference Lines

Reference lines may be defined, for instance, during system design: upon a requirements definition, a prototype or a simulation of the user interface; during software realization: upon an increment of the software version; or during use: upon a list of new or changed requirements.

The crucial point with the reference line concept is that neither the content nor form of intermediate products, nor the points of time at which they occur are predefined by the project model in advance**.** The participants determine the reference lines, i. e. the project states to be reached, so to speak, dynamically, as needed in the development process and in accordance with specific situations.

6. Methods and Tools for Software Development with Users

The version concept constitutes the translation into methodological terms of the necessity of revising a software system arising from the evolutionary property of requirements. The central component of a system version, the software version, must be capable of being meaningfully employed by the users as a tool in their daily work**.** At the same time, it must be designed to be modifiable in order to meet new and changed requirements.

New and changed requirements must be expected to result from the use of the software version, from changes in the technical environment and from an evaluation of the software itself. They cannot be planned in advance; they must, however, be given systematic consideration in methodological terms by a project model designed for long-lived software systems to be used in work processes

Conversely, the version concept may also be employed from the start for planning several development cycles, e.g. in cases where the requirements are too complex or unclear. We make a conscious distinction between system versions which are the resulting product of each development cycle, and prototypes which we employ exclusively for exploratory and experimental purposes.

The version concept is realized by two measures, a product-oriented and a process-oriented one.

The product-oriented measure involves structuring the products produced during one development cycle with respect to changeability and reuseability, in such a way as to enable them to be largely drawn upon during the following cycle. The process-oriented measure involves ensuring, from the very beginning, that users and developers understand system development as a cyclical process in which they perform specific roles, and not as a process terminating after the initial and definitive production and installation of a software system.

Participative system development is supported by methods for specific development tasks. Of particular interest are methods for establishing requirements and for designing human-computer-interaction.

In STEPS, the employment of methods is not tied up with distinct, predefined phases, but takes place in connection with the development tasks as they arise in successive cycles. In order to satisfy the needs of participative system development, methods must be geared to an incremental and cooperative working style. They may be methods specifically designed for these purposes as in /Keil-Slawik 89/ or generally available methods adapted to our specific needs (for example, we have made exccellent experiences using an adapted form of SADT).

Particular importance is attached to the technique of *prototyping*, which is applied for exploratory purposes during requirements analysis, and for experimental purposes during design of the user interface or software realization, for example. Prototyping enables feed-back to take place between users and developers, not only through reading and understanding documents, but also on the basis of experience gained in using the software system or individual components of it. Some experiences we have gained with prototyping in the PEtS-project have been reported in /Reisin, Wegge 89/.

Tools to support cooperative software development must be tailored to incremental work developing systems in versions. They must further provide a shared working-milieu for developers (and users, if desired) to facilitate the cooperation on shared results. An attempt to provide such tools in our technical environment is reported in /Grycyan, Kautz 89/.

7. STEPS to be Taken by People

One of my reasons for choosing STEPS as the name for our approach was my conviction that methods exist only as steps taken by people. That is, we deal with processes of method development, application and adaption to actual settings rather than with methods as products. Methods do not determine the quality of software products, but people involved in systematic design processes allow quality to emerge. Each project is unique and presents new challenges for cooperation and creativity.

What happens in software development is the responsibility of those participating. We see in STEPS a guidance for cooperative software development that may help others to take their own steps by learning from our experience and adapting our approach to their unique settings.

A methodical approach such as STEPS, however, can be fruitfully applied only in contexts that allow cooperative software development to take place. This implies permitting users to make decisions concerning their work with a high degree of autonomy and enabling them to do so by ensuring that they are properly qualified. It also means to make available the resources needed for carrying out cooperative

work. While this will lead to higher costs in the first stages of system development, there are serious reasons for hoping that these will pay off in terms of use quality and acceptance later on.

Fundamental prerequisities for being able to carry out cooperative work are human abilities in dialogue and mutual acceptance. There is no formula or method for acquiring those - just willingness and practice. Cooperative work comes with conflicts and misunderstandings, with criticism and difficulties in communication. All of us fail at some time. So let us take one another seriously, and try again - taking the next step together.

References

Andersen et al. 86: N.E. Andersen, F. Kensing, M. Lassen, J. Ludin, L. Mathiassen, A. Munk-Madsen, P. Sørgaard: *Professionel Systemudvikling*, Kopenhagen, 1986.

Bateson 80: G. Bateson: *Mind and Nature - A Necessary Unity*; Bantam Books 1980.

Bjerknes, Bratteteig 87a: G. Bjerknes, T. Bratteteig: *Perspectives on Description Tools and Techniques in System Development*. In: P. Docherty, K. Fuchs-Kittowski, P. Kolm, L. Mathiassen (Eds.): Systems Design for Human Development and Productivity: Participation and Beyond, North-Holland, Amsterdam, 1987.

Bjerknes, Bratteteig 87b: G. Bjerknes, T. Bratteteig: *Florence in Wonderland. System Development with Nurses.* In: G. Bjerknes, P. Ehn, M. Kyng (Eds.): *Computers and Democracy – A Scandinavian Challenge*, Avebury, Aldershot, England, 1987.

Bødker 86: S. Bødker: *User Interface Design*, Draft for publication in Utopia Report No. 15, Aarhus University, 1986.

Bråten 73: S. Bråten: *Model Monopoly and Communication: System Theoretical Notes on Democratization*. In: Acta Sociologica 1973, Vol. 16-No.2.

Bråten 78: S. Bråten: *System Research and Social Science*. In G. Klir (Ed.): Applied Systems Research: Recent Developments and Trends, New York, 1978.

Budde, Kuhlenkamp, Mathiassen, Züllighoven 85: R. Budde, K. Kuhlenkamp, L. Mathiassen, H. Züllighoven (Eds.): *Approaches to Prototyping*, Springer-Verlag, Berlin, 1985.

Budde, Floyd, Keil-Slawik, Züllighoven 89: R. Budde, C. Floyd, R. Keil-Slawik, H. Züllighoven (Eds.): *Software Development and Reality Construction* . To be published by Springer-Verlag, Berlin Heidelberg New York Tokyo, 1989.

Ehn 88: P. Ehn: *Work-Oriented Design of Computer Artifacts*, Almquist & Wiksell International, Stockholm, 1988.

Engeström 87: Y. Engeström: *Learning by Expanding*. Orienta-Konsultit Oy, Helsinki 1987.

Floyd 81: C. Floyd: *A Process-Oriented Approach to Software Development*. In: Systems Architecture. Proc. of the 6th European ACM Regional Conference, Westbury House, 1981.

Floyd, Keil 83: C. Floyd, R. Keil: *Adapting Software Development for Systems Design with the User*. In: U. Briefs, C. Ciborra, L. Schneider (Eds.): Systems Design For, With and By the Users, North-Holland, Amsterdam, 1983.

Floyd 85: C. Floyd: A Systematic Look at Prototyping; in: Budde, R., Kuhlenkamp, K., Mathiassen, L., Züllighoven, H. (Hrsg.): Approaches to Prototyping, Springer Verlag, Berlin, Heidelberg, New York, Tokio, 1985.

Floyd 86: C. Floyd: *A Comparative Evaluation of System Development Methods.* In: T.W. Olle, H.G. Sol, A.A. Verrijn-Stuart (Eds.): Information Systems Design Methodologies: Improving the Practice, North-Holland, Amsterdam, 1986.

Floyd 87: C. Floyd: *Outline of a Paradigm Change in Software Engineering*. In: G. Bjerknes, P. Ehn, M. Kyng (Eds.): Computers and Democracy – a Scandinavian Challenge, Gower Publishing Company Ltd., Aldershot, England, 1987.

Floyd 89: C. Floyd: *Softwareentwicklung als Realitätskonstruktion.* To be published in The Proceedings of the Conference *Software-Entwicklung – Konzepte, Erfahrungen, Perspektiven* held in Marburg/Lahn from 21-23 June, 1989.

Floyd, Mehl, Reisin, Schmidt, Wolf 89a: C. Floyd, M. Mehl, F.-M. Reisin, G. Schmidt, G. Wolf: *Zwischenbericht des Projektes PEtS.* Technische Universität Berlin, 1989.

Floyd, Mehl, Reisin, Schmidt, Wolf 89b: C. Floyd, M. Mehl, F.-M. Reisin, G. Schmidt, G. Wolf: *OUT OF SCANDINAVIA – Alternative Software Design and Development in Scandinavia – A Study of Methods, Concepts, Practical Projects and their Findings.* To be published in: Human-Computer Interaction, Autumn, 1989.

Gryczan, Kautz 89: G. Gryczan, K. Kautz: *Tool Support for Cooperative Software Development Tasks in STEPS.* To be published in the Proceedings of the 12th IRIS Conference to be held at Skagen, Denmark, August 13-16, 1989.

Holzkamp 78: K. Holzkamp: *Sinnliche Erkenntnis - Historischer Ursprung und gesellschaftliche Funktion der Wahrnehmung*. Frankfurt a.M., 1978.

Jackson 83: M. Jackson: *System Development*, Prentice-Hall International, Englewood Cliffs, New Jersey, 1983.

Keil-Slawik 89: R. Keil-Slawik: *Systemgestaltung mit Aufgabennetzen.* In: S. Maaß, H. Oberquelle (Hrsg.): Software-Ergonomie '89, B.G. Teubner, Stuttgart, 1989.

Lehmann 80: M.M. Lehmann: *Programs, Life Cycles and Laws of Software Evolution*, Proc. IEEE 68, 9 (1980).

Leontiev 78: A.N. Leontiev: *Activity, Consciousness, and Personality*. Englewood Cliffs, Prentice Hall, 1978.

Maturana, Varela, Uribe 74: H. Maturana, F. Varela, R. Uribe: *Autopoiesis, the Organization of Living Systems: Its Characterization and a Model*. In: Biosystems 5, 187, 1974.

Mumford 87: E. Mumford: *Sociotechnical Systems Design – Evolving Theory and Practice.* In: G. Bjerknes, P. Ehn, M. Kyng (Eds.): *Computers and Democracy – A Scandinavian Challenge*, Avebury, Aldershot, England, 1987.

Naur 74: P. Naur: *Concise Survey of Computer Methods*, Studentlitteratur, Lund, Sweden, 1974.

Naur 85: P. Naur: *Programming as Theory Building*, Microprocessing and Microprogramming 15 (1985) 253-261.

Nygaard 87: K. Nygaard: *Program Development as a Social Activity.* In: K. Fuchs-Kittowski, D. Gertenbach (Eds.): System Design for Human Development and Productivity: Participation and Beyond, Akademie der Wissenschaften der DDR, Berlin (GDR), 1987.

Parnas, Clements 85: D. Parnas, P. Clements: *A Rational Design Process: How and Why to Fake It*. In: H. Ehrig, C. Floyd, M. Nivat, J. Thatcher (Eds.): Proc. of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT), Vol. 2: Formal Methods and Software Development, Springer-Verlag, Berlin Heidelberg New York Tokyo, 1985.

Pasch 89: J. Pasch: *Mehr Selbstorganisation in Softwareentwicklungsprojekten*, Manuskript, Technische Universität Berlin, 1989.

Pask 75: G. Pask: *Conversation, Cognition and Learning*, Elsevier, Amsterdam Oxford New York, 1975.

Reisin, Wegge 89: F.-M. Reisin, Daniela Wegge: *On Experimental Prototyping in User-Oriented System Development.* To be published in the Proceedings of the 12th IRIS Conference to be held at Skagen, Denmark, August 13-16, 1989.

Ryle 63: G. Ryle: *The Concept of Mind,* Penguin Books, Harmondsworth, England, 1963.

UTOPIA 85: Graffiti 7: *The UTOPIA Project – An Alternative in Text and Images*, Summary Report, Stockholm, May 1985.

v. Foerster 60: H. von Foerster: *On Self-Organizing Systems and Their Environments.*In: M. Yovits, S. Cameron (Eds.): Self-Organiting Systems. Pergamon Press, London, 1960.

Vygotsky 62: L.S. Vygotsky: *Thougth and Language*. Cambridge, Mass., The MIT Press, 1962.

Weick 79: K.E. Weick: *The Social Psychology of Organizing*, Addison-Wesley 1979

Winograd, Flores 86: T. Winograd , F. Flores: *Understanding Computers and Cognition*, Ablex Publishing Corporation, Norwood, New Jersey, 1986.