

*Finn Kensing and Andreas Munk-Madsen*

## PID: STRUCTURE IN THE TOOLBOX

**In Scandinavia user participation arose in the context of action-oriented research with trade unions in the late 1970s. Both authors participated in such projects (also see the article by Clement et al. in this issue). The approach was oriented toward increasing the influence of workers and their unions over the development and use of technology at their workplaces.**

In this article, however, we argue from an epistemological standpoint that participatory design is needed to gradually build up the knowledge required for developing and using a new system.

There is a common explanation when an attempt to practice PD fails: "The users and the system developers did not understand each other." The statement is often followed by the recommendation of a specific technique or tool to remedy the situation. However, in our experience there is no foolproof method. System development projects fail in communication even though they use the most promising techniques.

In one project horizontal prototypes were used extensively during the time the requirements were defined. A horizontal prototype shows all intended functions, but they are not implemented in detail as required in the final system [16]. The intention was to ensure that the users understood what they accepted. The system had to undergo substantial changes, however, before it could be used [2].

In another case the users were unable to define system requirements at meetings with system developers. The system developers then made an elaborate vertical prototype and expected a response from the users. A vertical prototype offers a selection of functions implemented in their intended final form [16]. However, they did not receive any response from the users.

How do we account for these apparent paradoxes? It is difficult to find relevant explanations. Most papers and books deal specifically

with techniques and tools, not with underlying theories enabling us to discuss the context and the limitations of the techniques and the tools. Comparative surveys of methods [3, 18, 29] are usually thorough on details but lacking in explanatory theory.

In this article we suggest an answer to the communication paradoxes in terms of a model of user-developer communication. The model is based on theories dealing with system development as well as with communication. The model may help us understand why some approaches sometimes yield fruitful communication, while in other situations the same approaches turn out to be obstacles. The distinctions offered by the model may act as a catalogue—or toolbox—where system developers may find ideas appropriate for specific situations. We use the model to categorize communication methods and description tools in relation to their application area. Thus our model may form the basis of a contingency strategy, as proposed by Davis [11] and Boehm [4].

The model covers communication related to analysis and design (i.e., to defining requirements and creating solutions). It does not cover all user-developer communication. It excludes communication related to management and implementation.

### User-Developer Communication in System Development

We want to discuss possibilities and obstacles for successful communication in system development. Therefore we relate the communication processes to their results and to the

context in which they take place.

Describing the system development process, Clements and Parnas [8] state: "The most useful form of a process description will be in terms of work products." They proceed by describing the documents they would produce during a project's lifetime. We agree with them, although our concept of results is not confined to documents alone. We would also like to include the knowledge developed by the people involved as results.

What then are the results of the system development process? The final results are, of course, a system and a completed technical and organizational implementation process. Intermediate results are documents and knowledge obtained by the participants. Regardless of the development model—be it waterfall, spiral, incremental or parallel—these results form the basis of important decisions. These decisions deal with determining the system's level of sophistication, evaluating the usefulness of the system, freezing the requirements, and designing the system's internal structure.

Thus the goal of analysis and design activities is to produce documents and knowledge enabling decision-making with regard to the system and its environments. How can we produce these results (i.e., what kind of methods do we need?) That depends on the prerequisites for the development process, especially the limitations of user-developer communication. The following section presents a model of communication in order to answer this question.

## Communication Models

Communication is of course a key issue in collective activities such as system development. People with different backgrounds, education, training, and organizational roles exchange facts, opinions, and visions in order to inform, persuade, and maybe even threaten one another. How is communication possible in such a context.

We sketch two communication models relevant to understanding and designing user-developer communication: a traditional model and an alternative model. It is our opinion that many current tools and techniques rely heavily on the first model.

Current methods usually support written communication based on formalized languages, prototyping being the major exception. These methods rely on a communication model which can be described by a tube-for-communication metaphor. Communication is perceived as something created at one place (e.g., the developers' office), then carried through "a tube" to the receivers (e.g., the users). The tube could be some kind of written system description. This communication model takes for granted that successful communication is determined by the "sender's" ability to form a rigorous message. How is it, that the same message in the same form can be interpreted so differently by various "receivers"?

An alternative communication model focusing on the prerequisites of those involved in a communicative situation enables us to approach this question. When people communicate, the speaker's words may trigger a change of state in the listeners. According to Maturana and Varela [27] "communication depends on not what is transmitted, but on what happens to the person who receives it." The key criteria for successful communication within this model relates to the people involved, rather than to some kind of 'tube' between them. Thus, successful communication depends on the ability to establish situations in which mutual perturbations trigger changes in the state of those involved, which in turn lead to

structural congruence (social coupling) among communicating partners. Writing and speaking do not guarantee reading or listening—or, even more important—do not guarantee the establishing of the concepts and models intended by the 'sender'. Communication is created by people who interact.

Maturana and Varela state that a person's interaction domain is his or her domain of cognition. This implies that the kinds of activities in which we are involved delimit the kinds of knowledge we are able to develop. It further implies that the tools we apply in these activities delimit the kind of knowledge we are able to develop. The rejection of the tube-for-communication metaphor implies that developers and users must set aside much time for discussions and for joint activities. This is done at the expense of working alone and communicating solely in writing, which current methods primarily support. Techniques such as prototyping, mapping, future workshops, and metaphorical design (see section entitled "Tools and Techniques for Knowledge Development), are alternatives which support the development of social coupling, and thereby successful communication.

## A Model of User-Developer Communication

We want to be able to address such questions as: "Why did a specific project fail even though it contained many user-related activities?" "Which methods should be applied in specific system development situations?" "How do system developers ensure active user participation?"

In order to discuss these questions we have created a model of the communication between users and system developers. The model highlights important factors and relates them to one another. The factors are: the *results* of the system development process (including intermediate results); the participants' *prerequisites*, and *tools and techniques* for system description. The model is based on two distinctions—dealing with *three domains of discourse* and *two levels of knowledge*. The three domains of discourse are illustrated in

Figure 1.

Figure 1 illustrates the idea that design is bridge-building, since something new is created from two separate things. Design is based on two domains of discourse: the users' present work and the technological options. Here technology incorporates not only hardware and software, but also work organization. While this may seem strange, in this context we find it useful and acceptable to group these matters. Various organizational options, as well as several hardware and software options, should be considered and coordinated in order to fit together as well as possible. The result is a third domain of discourse: a new (or changed) computer system and changes in the content and the organization of the users' work.

These domains typically reflect the users' and developers' knowledge and understanding prior to entering the system development process. At the outset the users have some knowledge of their present work and of organizational options. The system developers have some knowledge of the technical options with regard to hardware and software. At the outset this is all they need to know.

Based on this distinction we state:

### **Thesis: The main domains of discourse**

*The main domains of discourse in design are:*

- \* *users' present work*
- \* *technological options*
- \* *new system*

*Knowledge of these domains must be developed and integrated in order for the design process to be a success.*

The second distinction is illustrated in Figure 2. It shows we need two levels of knowledge. We need abstract knowledge to get an overview of a domain of discourse and we need concrete experience in order to understand the abstract knowledge.

We combine the two distinctions into the model shown in Table 1. The model describes three main domains of discourse on two levels of abstraction. Altogether, we get six *areas of knowledge* in user-developer communication (Table 1).

**Table 1.** Six areas of knowledge in user-developer communication

	Users' present work	New system	Technological options
Abstract knowledge	Relevant structures on users' present work (2)	Visions and design proposals (5)	Overview of technological options (4)
Concrete experience	Concrete experience with users' present work (1)	Concrete experience with the new system (6)	Concrete experience with technological options (3)

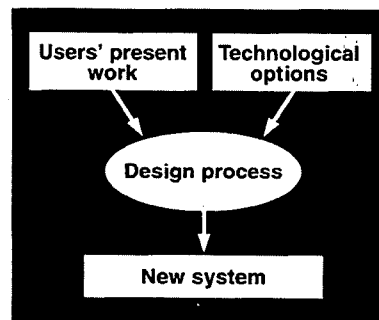
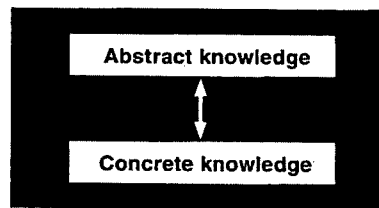
The numbering 1 to 6 in Table 1 does not reflect a time sequence, (i.e., we are not proposing a new waterfall model). The numbering is done for the purpose of convenient reference. Various methods propose different sequencing when dealing with the six areas. Normally we would expect some degree of iteration. However, a discussion of methods is beyond the scope of this article. The six areas of knowledge comprise a classification of system development tools and techniques. The following subsections discuss each area in more detail. The reader may wish to look ahead at Table 2 to see examples of tools and techniques in each area.

#### Concrete Experience with Users' Present Work

Developers need this area of knowledge [19]. They must have some feeling for the users' work in order to be able to understand and to produce structured descriptions or representations of this work (area 2). They cannot rely on users' talking about their work, and they cannot rely on a requirement specification. Developers must experience users in action.

If developers have no concrete experience with what is going on in the user organization and if they have no idea of the cultural potentials for change, they cannot judge the relevance of a structured description of the work. User representation in the design team does not overrule this statement.

The results of dealing with this area of knowledge may come in terms of experiencing differences in working styles, normal and stress situations, exceptions, power relations, and so forth. Results may also be the formation of a common language among users and developers.

**Figure 1.** Three domains of discourse in the design process**Figure 2.** Two levels of knowledge

#### Relevant Structures on Users' Present Work

A relevant structure defines a common and rigorous language in which users and developers can communicate. A structure is a model of the present situation in the user organization. The model is used to identify desired changes and to evaluate consequences of proposed designs. We refer to structures in the plural, as we cannot expect to capture the richness of the users' work in a single structure.

Which structures are relevant depends on the situation. Information flow is a structure offered by many methods. It is relevant when we want to automate existing data processing. A control model is a relevant structure when we want to discuss man-

agement information systems. A model showing the variety and inter-relationship of tasks carried out by individuals or a group during a typical working day is relevant when we want to discuss requirements for a new communication system.

#### Concrete Experience with Technological Options

If we want users to play an active role in system development we must provide them with technological options. This is done to stimulate their imagination and to enable them to better understand abstract descriptions of technical and organizational solutions.

The relevance of activities in this area is of course dependent on the users' present experience. Even if they are daily users of some kind of system, they might not have experienced the variety of existing hardware and software.

If we want designers to play an active role in designing the use of technology in organizations (although this is seldom an explicit goal, they often do this anyway) they must have organizational options. This is done to stimulate their organizational thinking and to enable them to understand the users' concrete experiences with, as well as their abstract descriptions of, organizational options.

#### Overview of Technological Options

This area of knowledge is the input of technical and organizational ideas into the design process. The system developers must be well informed about possibilities and limitations regarding hardware and software in order to justify their presence in the process. If nobody in the user organization has an overview of organiza-

tional options, then this subarea has to be developed during the design process to ensure that the new computer system and the new organization fit together.

#### Visions and Design Proposals

These descriptions are developed throughout a project's lifetime. Here too, it is a question of many structures, as one alone cannot capture the totality of a new computer system and its use. The structures document the actual progress of the project as it approaches the final result, forming the basis for renewed contracts, even if these may be informal. Therefore some of these descriptions must be understandable to the users.

Abstract descriptions are normally required as part of a system development project. These may be difficult for the users to understand, but they are necessary to the developers. We stress that in order for users to make decisions and assign priorities, they too need abstract descriptions to provide them with relevant structures of the new computer system, as well as of the organization in which it is to be implemented. These descriptions might very well differ from those needed by the developers.

#### Concrete Experience with the New System

The purpose of this area is to enable the users to understand abstract descriptions of the new system (area 5), and to let them experience how the new system meets their needs. The system developers also need concrete experience with the new system in order to check whether it fulfills the descriptions. In a specific project this area may already be covered through experience with technological options (area 3). This depends first and foremost on how radically the new system transcends current practice.

#### Theses Based on the Model

We now relate the model to the participants' prerequisites and we discuss which areas of knowledge each party must develop in order to facilitate genuine cooperation. The minimal starting point for a design process is actually rather narrow. Therefore it is the system developers'

**Table 2.** Tools and techniques for knowledge development

Tools and techniques	Areas of knowledge					
	1	2	3	4	5	6
Observations [24, 30]	1					
Interviewing users	1	2				
Self-registration [17]	1					
Developers doing users' work	1					
Videorecording [23, 30]	1					
Mock-ups [13, 14]	1					6
Think-aloud experiments [23]	1					6
Drawing rich pictures [7]	1	2				
Conceptual modelling [7]		2				
Culture analysis [5]	1	2				
Object-oriented analysis [9]		2			5	
Object-oriented design [10]					5	
Event lists [28]		2			5	
Entity-relationship diagrams [28]		2			5	
Wall graphs		2			5	
Mapping [25]		2			5	
Future workshop [21-23]		2			5	
Metaphorical design [23, 26]		2			5	
Dataflow diagrams [12]		2			5	
Language analysis [24, 30, 31]		2			5	
Card games [15]	1					6
Prototyping [1, 6, 16, 20]			3		5	6
Visits to other installations			3	4		
Literature study				4		
Study standard software			3	4		
Forum theater						6

responsibility to apply tools and techniques which allow the participants to acquire an understanding of areas in which they have little or no knowledge.

#### Thesis: Areas covered by the users.

*We can usually be sure that users cover area 1: Concrete experience with user work. We can usually expect nothing more.*

Obviously, users may be ignorant of technological options and the future system. However, it is not so obvious that they normally do not possess relevant structures or representations of their own work. The keyword here is "relevant."

Traditional structures, such as organization diagrams and descriptions of the formal division of labor are not necessarily relevant. They

may be insufficient when it comes to discussing inexpediciencies in the users' present work and requirements for new systems, since they do not necessarily reflect what can be observed in the organization. Relying on such descriptions has often led to solving the wrong problems.

**Thesis: Areas covered by the system developers.**

*We can usually be sure that system developers cover areas 3 and 4: technological options. We can usually expect nothing more.*

The first part of this thesis is rather obvious, or designers would have no role in the process. However, designers may also be challenged by the technological options. For example, when new development tools are applied, when simultaneous development of basic software and applications occurs, and when new standard software or hardware is introduced.

With regard to the second part of the thesis, the developers may of course have worked for the organization before, or they may have worked

for a similar organization. In that case they may have concrete experience as well as abstract knowledge about the users' present work. This would make things easier, but this is not something that can generally be taken for granted. Also when it comes to the new system, developers may have prior experience (e.g., from implementing standard systems). However, neglecting the characteristics of the specific organization will prevent the new computer system and the organization from fitting together.

**Thesis: Areas of knowledge to be acquired by the users through the development process.**

*It is the system developers' responsibility to apply tools and techniques allowing users to develop*

- \* relevant structures on users' present work (area 2),
- \* visions and design proposals (area 5),
- \* concrete experience with the new system (area 6).

The reasons for this thesis are the following: abstract descriptions of the new system (area 5) and relevant structures on users' present work (area 2) are needed when the users evaluate design proposals. As some part of the user organization must normally make a decision about accepting or rejecting proposals, the users' knowledge of these areas is indispensable.

In order for users to play a creative role in design they need abstract descriptions of their present work (area 2) as well as of the new system (area 5). Concrete experience is also needed, however, in order to understand abstract descriptions. Thus the users need concrete experience with the new system (area 6) before they can understand abstract descriptions of the new system (area 5).

**Thesis: Areas of knowledge to be acquired by the system developers through the development process.**

*It is the developers' responsibility to apply tools and techniques allowing them to develop*

- \* visions and design proposals (area 5),
- \* relevant structures on users' present work (area 2),
- \* concrete experience with users' present work (area 1),

## The ETHICS Approach

Enid Mumford  
CHESHIRE, ENGLAND

**E**ffective Technical and Human Implementation of Computer-based Systems—ETHICS—is a technique and also a philosophy that future users of new technical systems should be able to participate in the design process and help create systems that are humanistic and friendly as well as efficient and effective.

**ETHICS has three principal objectives:**

- to enable future users to play a major role in system design, and to assume responsibility for designing the work structure that surrounds the technology. This involves a learning process and a set of simple diagnostic and socio-technical design tools.
- to ensure that new systems are acceptable to users because they both increase user efficiency and job satisfaction.
- to assist users to become increasingly competent in the management of their own organizational change, so that this becomes a shared activity with the technical specialists and reduces the demands on scarce technical resources.

The methodology is not necessarily aimed at producing a computer-based solution, as the emphasis is on obtaining the right balance between the social and technical aspects of the complete system. ETHICS incorporates the joint philosophies of participation and socio-technical design.

It assists user design groups to create a decision structure that incorporates all interested groups affected by the new system; a process that enables the de-

sign task to be smoothly carried forward from identification of need to change, to successful operation of the new system. It also sets an agenda that enables business efficiency and employee job satisfaction improvements to be considered in parallel and given equal weight.

**ETHICS incorporates the following diagnostic and design tools:**

- a framework to assist the identification of mission, key tasks, important constraints and factors critical to effective operation,
- a variance analysis tool to assist the identification of systemic and operational problems and problem areas,
- a questionnaire to measure job satisfaction,
- a framework to identify what is likely to change in the internal and external environments, and
- a set of guidelines for individual and group work design.

The approach is currently used in three ways. First, it is used for the task for which it was originally designed—to help future users diagnose their needs and problems, setting nontechnical objectives for the system and restructuring their work situation. In this form it has been used by shop floor and office workers, sales staff and nurses.

ETHICS is also assisting managers to define their information needs prior to introducing a semantic logic-based executive information system. Here, a simplified form of ETHICS, called QUICKETHICS—Quality Information from Considered Knowledge—has been developed.

Finally, the method is employed as a general problem-solving tool to enable groups to systematically analyze needs and problems with a view to improving performance. **E**

Mumford developed the ETHICS approach.

\* *concrete experience with the new system (area 6).*

It goes without saying that the developers must understand abstract descriptions of the new system (area 5) since they are major intermediate results. Relevant structures on users' present work (area 2) must be understood in order to identify and evaluate desirable changes.

The developers must have concrete experience with users' present work (area 1) in order to understand and produce descriptions of relevant structures on the users' present work. System developers who have developed this area of knowledge have a better background for communicating with the users, as they are able to refer to and understand references to concrete events in the users' organization.

Finally, the developers must have concrete experience with the new system (area 6) in order to be able to test and evaluate the products of their own work.

We conclude this section with the observation that our theory recognizes that all areas of knowledge must be dealt with in any normal system development process. The toolbox necessary for this work is discussed briefly in the following section.

### Tools and Techniques for Knowledge Development

Our model of user-developer communication can be used to define a toolbox for tools and techniques to facilitate this communication. The toolbox presented in Table 2 consists of 6 sections, one for each of the areas of knowledge discussed in the previous section. It accentuates the differences between the purposes of the tools and techniques, even though some fit into more sections.

The use of any tool or technique must be adapted to the particular conditions of each system development process. In particular, the users' prior experience with the tools and techniques must be considered. Some tools and techniques are almost always used successfully by developers and users working together. Others should rather be part of an extended process, in which developers

## STEPS—A Methodical Approach to PD

Christiane Floyd  
UNIVERSITY OF HAMBURG

**S**oftware Technology for Evolutionary Participatory System Design—STEPS—is a methodological framework for developing interactive application systems. It guides developers and users in carrying out their cooperation as well as supports the design of computer-based work. STEPS emphasizes the development process and intertwines development with use.

The method has been tried successfully in participatory development of an information system called the PETS project, which is designed for handling archives of union contracts in Germany. This participatory project resulted in a working system that is still in use today.


STEPS considers the anticipation of use an inherent part of the development. Whenever software systems—as tools or media—are to be fitted into work processes, development consists in unfolding the problem as well as in elaborating a solution. The 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 problem.

Therefore, software development becomes a learning process for both developers and users. In STEPS, *evolution* refers to the emergence of insights into the functionality and the potential use of software. *Participation* refers to the strategy of mutual learning. Those participating in a software development project are creating a product, and at the same time, the development process itself. These two complementary dimensions are reflected by product-oriented and process-oriented activities. STEPS provides guidance to developers and users in both dimensions.

This approach relies on perspectivity for gaining insight—making perspectives explicit and allowing them to interact. The use perspective held by those who interact with software is distinct from the development perspective held by software developers. Furthermore, they both arise in a variety of views related to functional roles, collective interests, and individual tastes. Multiperspectivity is a fundamental prerequisite for cooperative work which rests on perspective-based

modelling and evaluation when using software. Software requirements are related to the context of user work processes as a whole. They cannot be completely fixed in advance as they evolve because of changes in the organization. Also, the software system itself gives rise to new requirements.

Thus, STEPS is an approach that accommodates various forms of prototyping and portraying system development in cycles of version production, application and revision. A system version consists of software and its defining documents, supplemented by guidelines for the organization of work to be supported. The interplay between each software version and the associated reorganization of user work is anticipated in the cooperative design and evaluated in the revision step. Thus, it supports mutual learning by developers and users by carefully establishing and coordinating processes of cooperation.

STEPS rests on a project model suitable for PD. This model is cyclical, all development steps and products being subject to revision. It combines software production and application, visualizing the tasks of both developers and users. It allows the choice of a situation-specific strategy in the actual project. It relies on a minimum of predefined intermediate products, thus allowing freedom for choosing them as needed. It provides for temporal flexibility in cooperation. It incorporates the dynamic coordination of the ongoing project through establishment and reference lines. STEPS embodies a human-centered notion of quality and creates a platform for discussing criteria on how to design computer-supported work and facilitates the emergence of quality through cooperative design. 

#### References

- Floyd, C., Reisin, F.-M., Schmidt, G. STEPS to Software Development with Users. C. Ghezzi, J.A. McDermid, Eds.: ESEC '88, Lecture Notes in Computer Science Nr. 387, Springer-Verlag, 1989, pp. 48–64.
- Floyd, C., Mehl, M., Reisin, F.-M., Schmidt, G., Wolf, G. Projekt PETS: Partizipative Entwicklung transparenter Software fuer EDV-gest. tztze Arbeits-pl Arbeit, Gesundheit und Soziales des Landes Nordrhein-Westfalen, Technical University of Berlin, 1990.
- Floyd, C., Zilligoven, H., Budde, R., Kell-Slawik, R., Eds. Software Development and Reality Construction. Springer Verlag, 1992.

gather information in cooperation with users, produce descriptions in isolation, and finally present, discuss, and alter the descriptions again together with the users. Techniques such as future workshops and mock-ups belong to the first category, while object-oriented analysis and conceptual modeling belong to the latter.

A presentation of the tools and techniques chosen to illustrate the use of the toolbox is beyond the scope of this article. The interested reader may find additional information in the references indicated in Table 2.

### Conclusion

We find the model listing areas of knowledge in Table 1 useful for a classification of tools and techniques. Developers may find this classification helpful when planning a project.

We also find theses discussed in the previous section, "Theses Based on the Model" useful in explaining why projects run into trouble. This may be related to power games in the user organization or to other factors which are most often out of the developers' control. However based on our own research [2], we claim that far too often problems in real-life projects are caused by developers using inadequate tools and techniques.

We can now explain apparent paradoxes such as: "Horizontal prototypes are insufficient" [20] and "Prototypes do not substitute analysis" [1]. A horizontal prototype does not really give users an experience with the future system. It is more like an abstract system description: the menu hierarchy implemented on edp-hardware. Thus, inexperienced users will not obtain sufficient understanding of the system's functions. Vertical prototypes used successfully might solve the problem. On the other hand, prototyping diverts attention from general questions such as: Do we need a new computer system? To answer this question knowledge area 2 in Table 1 must be dealt with. Analysis techniques must also be used.

Table 2 not only indicates the areas of knowledge, in which the various tools and techniques are ade-

quate but at the same time also highlights the areas in which they are inadequate. Conclusions concerning the more established tools and techniques such as dataflow diagrams are interesting. One of many conclusions we may draw from Table 2 is that all traditional system development methods deal only with areas 2 and 5, resulting in abstract descriptions. Thus, by themselves they are insufficient as guidelines for the entire system development process. They must be supplemented by techniques giving concrete experiences of user work and computer technology.

### Acknowledgments

We thank Kaj Grønbæk, Jesper Holck, Lucy Suchman, Randy Trigg and Terry Winograd for comments on earlier drafts. Also we would like to thank Jennifer Oakley for a serious language tune-up. □

### References

1. Andersen, N.E. Brug af prototyper (in Danish) (Using Prototypes), Datacentralen, 1987.
2. Andersen, N.E. et al. *Professional Systems Development*. Prentice-Hall, Englewood Cliffs, N.J., 1990.
3. Blank et al. *Software Engineering: Methods and Techniques*. Wiley-Interscience, 1983.
4. Boehm, B. A spiral model of software development and enhancement. *Computer* (May 1988).
5. Bødker, K. and Pedersen, J.S. Workplace cultures—Looking at artifacts, symbols, and practice. In [19].
6. Bødker, S. and Grønbæk, K. *Design in action—From prototyping by demonstration to cooperative prototyping*. In [19].
7. Checkland, P. *Systems Thinking, Systems Practice*. Wiley, New York, 1981.
8. Clements, P.C. and Parnas, D.L. A rational design process: How and why to fake it. In *Proceedings of the International Joint Conference on Theory and Practice of Software Development*. Springer Verlag, New York, 1985.
9. Coad, P. and Yourdon, E. *Object-Oriented Analysis*. Prentice Hall, Englewood Cliffs, N.J., 1991.
10. Coad, P. and Yourdon, E. *Object-Oriented Design*. Prentice Hall, Englewood Cliffs, N.J., 1991.
11. Davis, G.B. Strategies for information requirements determination. *IBM Syst. J.* 21 (1982), 4–30.
12. DeMarco, T. *Structured Analysis and Systems Specification*. Yourdon Press, 1978.
13. Ehn, P. Work-oriented design of computer artifacts. *Arbetslivscentrum* 1988.
14. Ehn, P. and Kyng, M. *Cardboard computers—Mocking-it-up or hands-on the future*. In [19].
15. Ehn, P. and Sjögren, D. *From system descriptions to scripts for action*. In [19].
16. Floyd, C. *A Systematic Look at Prototyping*. In *Approaches to Prototyping*. R. Budde et al, Eds., Springer Verlag, New York, 1984.
17. Foged, J. et al. *Håndbog om Klubarbejde, edb-projekter og nye arbejdsformer*. (in Danish). TIK-TAK projektet, Århus University, Denmark, 1987.
18. Freeman, P. and Wasserman, A.I. *Software Development Methodologies and Ada*. DoD, 1982.
19. Greenbaum, J. and Kyng, M. Eds., *Design at Work: Cooperative Design of Computer Systems*. Lawrence Erlbaum, 1991.
20. Grønbæk, K. *Rapid Prototyping with Fourth Generation Systems—An Empirical Study*. DAIMI PB-270, Århus University, Denmark, 1988.
21. Junk, R. and Müller, N. *Future Workshops—How to Create Desirable Futures*. Institute for Social Invention, London, 1987.
22. Kensing, F. *Generation of Visions in Systems Development*. In *Systems Design for Human Development and Productivity*. P. Docherty et al, Eds., *Proceedings of the IFIP TC 9/WDG 9.1 Working Conference*. North-Holland, Amsterdam, 1987.
23. Kensing, F. and Madsen, K.H. *Generating visions—Future workshops and Metaphorical Design*. In [19].
24. Kensing, F. and Winograd, T. *The language/action approach to design of computer support for cooperative work—A preliminary study in work mapping*. In *Collaborative Work, Social Communications and Information Systems*, R.K. Stamper et al., Eds., *Proceedings of the IFIP TC8 Working Conference*. North-Holland, Amsterdam, 1991.
25. Lanzara, G.F. and Mathiassen, L. Mapping situations within a system development project. *Inf. Manage.* 8, 1.
26. Madsen, K.H. Breakthrough by breakdown—Metaphors and structured domains. DAIMI PB-243, Århus University, Denmark, 1988.
27. Maturana, H.R. and Varela, F.J. *The Tree of Knowledge—The Biological Roots*