On Participatory Design and User Involvement as Topics in Computing Education: A Contribution to a Curriculum Debate

Karlheinz Kautz
Norwegian Computing Center, P.O. Box 114 Blindern
N-0314 Oslo, Norway
Karl.Kautz@nr.no

ABSTRACT
The topic of participatory design has not yet been of central interest in the context of formal education for computing professionals. This paper addresses this issue and argues why an up-to-date education should include more human-centred approaches like participatory design. It takes its starting point in the ongoing curriculum debate and discusses how mathematical-, and engineering-based approaches and traditional system development training contribute to education in computer science and system development. It argues that all of these approaches have shortcomings as they each relate to a merely technical-oriented paradigm, ignoring vital aspects of computing, namely organisational, social, and political ones. It is therefore concluded that participatory design as an approach which deals with these issues should be part of a comprehensive computing education. The article is meant to provide the ground for the next step in a debate about participatory design and education, a discussion about how to integrate participatory design issues in current and future curricula and how to teach them.

KEYWORDS: Computing education, Perspectives on software development, user involvement

INTRODUCTION

Participatory design, user involvement, and issues of social computing are gaining more and more attention as topics in the field of computing. The biannual conferences on participatory design ([1], [2]) are one sign, two special numbers of the Communications of the ACM ([3], [4]) another one.

However, these topics are rarely addressed in the context of formal education in universities and schools, either by the participatory design community or by a broader curriculum debate in computer science and system development.

This article tries to make up for this omission. As a first step for getting a discussion going and for gathering the different initiatives, it will argue *why* it is important to integrate these topics into current and future curricula and why to teach them. As such the article is meant to provide the ground for the next step, for further arguments on *how* to do the integration and teaching.

In *PDC'94: Proceedings of the Participatory Design Conference.* R. Trigg, S.I. Anderson, and E.A. Dykstra-Erickson (Eds.). Chapel Hill NC USA, 27-28 October 1994. Computer Professionals for Social Responsibility, P.O. Box 717, Palo Alto CA 94302-0717 USA, cpsr@cpsr.org.

The curriculum debate in the United States serves as a starting point. In 1989, an ACM Task Force on the Core of Computer Science published its recommendations placing an emphasis on software development [5]. These recommendations were deeply rooted in a traditional perspective of the discipline with mathematics and engineering as its fundamentals. The committee proposed to change the name of the discipline into "Science of Computing" and recommended only gentle reforms.

Accordingly, the curriculum contains the traditional components, algorithms and data structures, programming languages, architecture, numerical and symbolic computing, software methodology and engineering, databases and information retrieval, artificial intelligence and robotics, and as the only tribute to a development of the discipline, human-computer communication. No statements, however, are made about the concrete amount of mathematics and engineering within the courses. Nor is there a topic on "computers and society" in the curriculum.

A Joint Curriculum Task Force formed by the ACM and the IEEE Computer society published their recommendations in 1991 [6], for more comments see also [7] and [8]. The proposal has the same basic structure as the one of the earlier task forces. However it includes a component on "social and professional context" in which undergraduates are to be taught the basic cultural, social, legal, and ethical issues inherent in the computing discipline. No explicit mention is made of wider aspects concerning organisational issues, participatory design or user involvement. A similar debate can be found in the German-speaking countries (see [9-12]), where we however find an explicit proposal for a software ergonomics education incorporating some of these topics [13].

In the remainder of this article a closer look will first be taken at the demands for a mathematical-, and engineering-based education. Then traditional system development as another basis for software development will be discussed. On the background of the identified deficiencies the concluding argument why participatory design issues should be of an education for all computing professionals is put forward.

THE DEMAND FOR A MATHEMATICS-BASED EDUCATION

Dijkstra is a representative of those demanding a mathematics-based education. He contributes to the curriculum discussion 1989 with his paper entitled "On the Cruelty of Really Teaching Computer Science" (in [15]). From his point of view, computers are technical artifacts with a complexity which exceeds all other technical artifacts so far. However, the only thing he sees a computer can do is to manipulate symbols and to produce results of such manipulations. A program is an abstract symbol manipulator which can be turned into a concrete symbol manipulator by supplying a computer to it. Dijkstra presents benefits of seeing programs as formulas and arrives at the conclusion that computer science is concerned with the interplay between mechanised and human symbol manipulation. This is what, according to him, is usually referred to as "computing" or "programming".

Hence, dealing adequately with the complexity of computers should be grounded in a mathematical-logical orientation of the basic education in computer science. The aim of such an education is to unfold the ability of writing correct programs, which means to transform given specifications into correct executable formulas.

Dijkstra admits that there has to be more in an education of computer science than the formal derivation of programs. Computers are a radical novelty. Therefore the use of well-known terms from other technical fields like software engineering, software maintenance, software tools, and programmers' work bench are misleading and give the impression that we are dealing with a known and easy to control technology. He warns of an unreflected use of such terms and demands that we stop referring to parts of programs or devices in anthropological terminology. But he makes a clear distinction between what he calls the "correctness problem" and the "pleasantness problem." The former deals with the question of how to design an artifact that meets its specification, whereas the latter comprises all those issues that cannot be handled by mathematical formalisms. Dijkstra does not give the impression that these issues are important enough to have a legitimate place in a basic computer science education.

Gries [15] takes a similar stand. On the basis of a report of the Computer Science and Technology Board in the US, he also argues that a more rigorous use of mathematical techniques including formal methods and mathematical proofs will help to improve the low quality of software.

Computer science as a discipline lacks professionalism and does not pursue the development of professional standards with the same energy as is usual in other engineering disciplines. From his point of view, the field relies far too much on intuition and guessing. What is needed to tackle the problems is a strong emphasis on basic mathematical skills that support dealing with algorithmic concepts.

The pure mathematical approach to software development has been widely criticised. Among others, Winograd in his answer to Dijkstra's paper argues that computers are not just dealing with mathematical objects (in [14]). It might be right that they only manipulate symbols, but they do this as a means to an end. Computers are devices that fulfill certain functions within human activities. Software development does not aim at the perfect and optimal solution, but at the construction of reliable and reasonable ones.

A second flaw in the mathematics-based argumentation is seen in the idealised view of programming as transforming a given specification into an executable program as the main task of computing professionals. This view seems to ignore the fact that there are big software packages in use which have not been formally specified. The problem does not seem to be dealing with formalism concerning existing specifications and programs, but how to gain the specifications, both informal and formal ones.

Winograd concludes that it would be foolish to ignore the value of the abstract mathematical skills Dijkstra advocates, but it would be even more foolish to indulge the fantasy that they offer some magic that allows students to escape the hard work of learning about real computing.

He supports an education which enables future computer professionals in a good engineering tradition to specify and design hardware and software devices in such a way that they will work effectively.

THE DEMAND FOR AN ENGINEERING-BASED EDUCATION

This leads to the demand for an engineering-based education which understands computer science, (information) systems, and software development as engineering disciplines. Lately the term "information systems engineering" has emerged (see [16]). The term "software engineering" was coined 25 years ago on the background of the so-called software crisis which was manifested by software systems which were too expensive, contained numerous errors, were not ready in time, frequently did not fulfill their users' requirements and often did not lead to the expected economic savings.

A more engineering-based approach which is based on the types of theoretical foundations and practical disciplines that are traditional in the established branches of engineering is seen as a way out of this crisis. Macro and Buxton (in [17]) define software engineering as "the establishment and use of sound engineering principles and good management practice ... in order to obtain ... software, that is of high quality ...".

It is obvious that the engineering approach aims at technical solutions and it is interesting to note that the approach was launched in a situation where the software systems to be developed were thought to be mainly used in technical contexts like controlling aircraft or telephone circuits, or to support engineers, for example, to design buildings, or to construct well-understood solutions for well-defined problems like payroll accounting. It seems that this assumption is still shared by many supporters of the engineering perspective today.

Parnas [18] relates computing explicitly to engineering and puts forward the following line of argumentation. He argues that most computing science graduates end up working in engineering jobs, thus they as such work as engineers as they are constructing technical artifacts. However in contrast to traditional engineers, students in computing do not learn fundamental engineering principles like systematic planning, analysis, documentation, and validation.

The reasons for these deficiencies in computing education are seen in the more random-like development of the early curricula in the 60's. Mathematical and engineering-based topics were compressed into quick shallow courses, the biggest part of the curriculum was formed by subjects which were understood as the "good stuff", this mirrored the then actual research interests like programming languages and accompanying language compilers. As a result, Parnas today sees theoretical computer scientists who seem to lack an appreciation for mature mathematics and practitioners who lack an appreciation for the essentials of professional engineering.

As a consequence he proposes a return to an approach in education which emphasises the classical fundamentals of engineering. Parnas is aware of the fact that some might find this curriculum old-fashioned, but he argues that it will allow for a flexibility and a lifetime of learning of new development, which is necessary in such a dynamic and fast changing field as computer science.

The undergraduate education should mainly be based on mathematics and engineering, supplemented with basic knowledge in physics and chemistry. Elementary computing science should be restricted to principles of structured programming, analysis and design of algorithms and data structures, technical documentation, systems' architecture. Parnas and Dijkstra agree that practical programming with concrete languages on concrete machines should not be part of an undergraduate education.

One can hardly disagree that knowledge of engineering and technical principles, in technical design, in programming techniques, in documentation, as well as in quality control procedures have to be part of the equipment of a system developer. Without the basic technical know-how, no trade can produce quality products. As Winograd puts it, an engineering education needs a grounding in the experience of the profession. Experience can partly be passed on by examples, but has its main source in practice. He therefore requests a more practical oriented education which goes beyond the building and testing of small programming exercises. It should include working with large-scale systems and the design considerations that come from their embedding in situations of use.

But within the engineering perspective emphasis tends to be merely on the technical construction of software. There seems to be a belief that requirements of software components can be described definitely and completely. Engineers seem to think that it is this kind of facts they can base their work on and that they have nothing to do with work organisation, work activities, and requirements analysis.

There is also the opinion that conformity between specification and code is the adequate criteria for assessment of software and that only minor interaction between developers and clients and future users is necessary after an initial agreement about the specification between these groups has been reached. These, by the way, are positions which, although they disagree about the importance of formal methods, are supported by both representatives of the classical mathematical and members of the traditional engineering school.

This approach largely excludes the problems arising from determining requirements and proposing functional specifications. This is not that much a problem in technical areas where requirements are frequently fixed. But system development takes place in a context and computer and information technology frequently is embedded in organisations and changes of the requirements are often triggered by changes in the organisation or have their basis in factors which are not engineering issues, like misunderstandings and poor communication between, and within, clients' and developers' communities. What is needed is an understanding by computer professionals who do not only know engineering, but also can deal with this kind of problem.

TRADITONAL SYSTEM DEVELOPMENT EDUCATION

While the engineering perspective explicitly excludes the organisational context, a look at the field of information systems development and its relationship to computing shows that the emphasis here is on using the devices for processing business information in organisations rather than considering the devices as a subject of concern on its own. The focus of interest is extended to analysing the business, establishing requirements, specifying functions, and dealing with people as users in organisations.

An International organisation, the International Federation of Information Processing, represents those active in this field. Already in 1968, the IFIP Technical Committee for Education initiated a working group to prepare a curriculum. The aim was primarily to address system analysis and design professionals. But there was also the foresight that there would be a need of education for those whose primary interests are in business and administration and to whom computer technology is only one management tool among others [19]. The topic of information system development and use attracted business schools. In fact, in the US according to Davis there still is a clear distinction between the field "information systems" of which 80% is taught in business schools, and "computer science" (see [19]).

Information systems development and software engineering have a number of things in common when it comes to software development: both have to deal with project management and organisation as well as with methods and tools for the development of software. They have different starting points: information systems development in system analysis and system design focusing on requirements and functional specifications, software engineering in coding and technical design, stressing the technical activities.

A look into information systems text books and software engineering text books shows that the same methods for the early activities in software development projects are presented. The difference is that software engineering books still tend to give technical examples. Pressmann [20], for example uses the development of a home security system throughout his whole text. In information systems literature, on the other hand, the examples deal with applications in which people in organisations explicitly are involved. Olle et al. [21], for example, present a flight reservation system to demonstrate system development methodologies.

The two fields overlap each other. Iivari [22] goes as far as identifying software engineering as one of the 'schools' within information systems development. The boundaries between the fields are blurred. To a certain extent it makes no sense to make a distinction between them at all. It looks as if the software engineering community at least to a certain degree has recognised the significance that other than mere technical issues have for software development. On the other hand the information system community seems to acknowledge the benefits of some engineering discipline.

Then, however, taking seriously that software components are not merely technical artifacts, but parts of organisations, means to include issues like establishing requirements, specifying functions, and in particular, dealing with users and user organisations in a computing studies curriculum independently from a particular perspective.

This does not mean that only special methods should be taught. Emphasis should be on underlying principles with the use of certain methods as practical examples. Every professional should have this knowledge, regardless of whether he or she gained education at a university or a business school and there are already many places which offer this kind of education.

However, in a conventional perspective, organisations are seen as unitary structures with a manifest and rational, and for the most part hierarchical organisational reality. This reality consists of objects, properties and processes that are directly observable. Thus, requirements specifications and design descriptions are considered as clear-cut documents which are as objective as possible. The resolution of polemical issues within the organisation is seen as a prerogative of management and not normally within the domain of the system developers.

Their role is to be neutral experts in technology, tools, and methods of system analysis and design, and project management. Using these supporting means is seen to make system development more rational, placing less reliance on human intuition. Management provides the objectives and dictates the ends for the system development projects. The system developers take the objectives and turn them into a constructed product. Users operate or interact with information technology to achieve organisational objectives.

In this view social issues are, if at all, considered in a very simple way and social phenomena are tackled in terms of static unilateral cause-effect laws. Computing professionals of all three schools presented so far share this with traditional organisational and management theorists. Management as a discipline of communication has under the influence of Taylorism been interpreted as a system of formal directives for action and formal descriptions of activities and tasks. In this view, communication is context-free and unilateral: management gives directives, staff carry them out.

This approach has been successfully used when formalised application areas are concerned. But it leads to some limitations when support for less formalised activities and structures and more dynamic organisations is aimed at. Specifications then often comprise only out-dated requirements and do not capture the "real" working practice in an organisation. Thus, the information technology applied does not support the work tasks of its users and further, quick and flexible adaptation to frequently changing organisational structures and tasks is hardly possible. The literature is full of stories reporting information technology failure.

This is what Hirschheim and Klein [23] call the orthodox approach to information system development and has much in common with what Floyd [24] calls the traditional, product-oriented perspective on software engineering. It takes note of the existence of an organisational context. However it is very close to the engineering perspective as the general orientation is towards technical issues including some behavioural consequences. But it is oriented not towards social issues as a main subject of concern. Thus including organisational issues in a curriculum for computing professionals is a step in the right direction, but it is not sufficient.

BEYOND THE TRADITIONAL APPROACHES TO COMPUTING EDUCATION - THE DEMAND FOR TEACHING PARTICIPATORY DESIGN

System development, according to Andersen et al. [25], consists of all those activities that aim at changing an organisation through the use of computer technology. Software development as part of system development is not just a process of technical change, but also, or first and foremost, means organisational development. Hirschheim et al. [26] argue that system development which aims at the use of computer technology is a social process, which relies on technology.

Wastell [27] and Bjerknes [28] describe this social process as a dialectical reality of mutual reciprocating influences and contradictions in which organisations are seen as pluralistic structures. There, many points of view and interests exist. This makes system development a complex, social phenomena. The political dimension is, for example, stressed by Marcus [29] and by Franz and Robey [30]. Solutions and prescriptions that oversimplify the actual reality will hardly lead to success.

System developers should try to understand the organisational, social, and political context of workplaces. They should not abstract away this context to pure information processing aspects as this is done by traditional approaches. There, system developers use structured analysis and design methods which allow them only to analyse and describe the technical features of organisations (cf. [29]). The failure of numerous system development endeavours due to neglecting these aspects is frequently reported (see, for example, [31], [32]).

As a consequence alternative approaches have emerged. The system developers in these approaches are not neutral or objective, they take over responsibility for what happens in an organisation when information technology is to be introduced. The approaches may be distinguished by the different roles the system developers may play in development projects. Avison and Wood-Harper [33] make a distinction between system developers as facilitators, as agents for social progress, and as emancipators. Similar distinctions can be found in Hirschheim and Klein [23] and Dahlbom and Mathiassen [34]. Such a differentiation, however, is not relevant in the context here.

What is relevant here, is what the approaches have in common: they all move the centre of interest away from organisations as abstract structures. The centre of interest is moved towards the everyday working practice of people within organisations. The development process is interpreted as a process of getting the users to understand, formulate, and define their problems and needs, instead of letting managers or consultants formulate a number of fixed problems. The need for user involvement is acknowledged. In line with Floyd [24] (see also [35]), Denning [36] sees a new paradigm for software development here. This paradigm has become known by the terms user-centred design or participatory design. Denning considers this paradigm to be well suited for the development of software, as it is required today, software that satisfies its users and supports their work. He argues that in this respect it is superior to formal approaches to software development.

Knowledge about, and experience with, user involvement and participatory design as well as accompanying techniques exist. As the material is well-documented (see [1]-[3], [37]-[41]) it can also be taught. It has to be taught as teaching it - together with all the other topics related to system development - means to offer an education for computing professionals which does not omit important parts, but which comprises all vital aspects of system development. This kind of system development education should be firmly established in any computing curriculum.

SUMMARY

In summary, different approaches to computing education have been presented in this article. The mathematical one, the engineering one, and the system development approach supplement each other and a modern curriculum must comprise all of them. But it is not sufficient to stick to the traditional, merely technocentric approaches. A modern education also has to include human-centred approaches like participatory design. This is necessary because computing professionals have to be prepared to meet not only technical, but also organisational, social, and political challenges, and it is possible because the existing knowledge is fairly well documented. Further discussions should now address how these issues can be integrated in computing curricula and how they actually can be taught. For this purpose it will be useful to collect and compare existing teaching experience.

REFERENCES

1. P. Czyzewski, J. Johnson (eds.), Proceedings of the Participatory Design Conference '90, Seattle, Palo Alto, CA, Computer Professionals for Social Responsibility, 1990.
2. M. J. Muller, S. Kuhn, J. A. Meskill (eds.), Proceedings of the Participatory Design Conference 1992, MIT, Cambridge, US, 1992.
3. Communications of the ACM, Special Issue on Participatory Design, Vol. 36, No. 4, June 1993.
4. Communications of the ACM, Special Issue on Social Computing, Vol. 37, No. 1, January 1994.
5. P. J. Denning, D. E. Comer, D. Gries, M. C. Mulder, A. B. Tucker, A. J. Turner, P. R. Young, Computing as a Discipline, in Communications of the ACM, Vol. 32, No. 1, pp. 9-23, 1989.
6. A. B. Tucker, B. H. Barnes, Flexible Design: A Summary of Computing Curricula 1991, in IEEE Computer, Vol. 24, pp. 56-66, November 1991.
7. P. J. Denning, Educating the new Engineer, in Communications of the ACM, Vol. 35, No. 12, pp. 83-97, 1992.
8. J. Hartmanis, Computing the Future, in Communications of the ACM, Vol. 35, No. 11, pp. 30-40, 1992.
9. L. Bonsiepen, W. Coy, Eine Curriculumdebatte, in Informatik Spektrum, Vol. 15, pp. 323-325, 1992.
10. M. Broy, Zur Aus- und Weiterbildung im Bereich der ingenieurmässigen System- und Programmentwicklung, in Informatik Spektrum, Vol. 16, No.1, pp. 31-33, 1993.
11. J.-A. Müller, Kommt die Entwicklung betrieblicher DV-Anwendungssysteme ohne Systems Engineering aus?, in Informatik Spektrum, Vol. 16, No. 3, pp. 167-169, 1993.
12. S. Wendt, Defizite im Software Engineering, in Informatik Spektrum, Vol. 16, No.1, pp. 34-38, 1993.
13. S. Maass, D. Ackermann, W. Dzida, P. Gorny, H. Oberquelle, K.-H. Rödiger, W. Rupietta, N. Streitz, Software-Ergonomie-Ausbildung in Informatik-Studien-

gängen bundesdeutscher Universitäten, in Informatik Spektrum, Vol. 16, No.1, pp. 25-30, 1993.
14. P. J. Denning (ed.), A debate on Teaching Computer Science, in Communications of the ACM, Vol. 32, No. 12, pp. 1397-1414, 1989.
15. D. Gries, Teaching Calculation and Discrimination: A more effective Curriculum, in Communications of the ACM, Vol. 34, No. 3, pp. 44-55, 1991.
16. A. Sølvberg, D. C. H. Kung, Information Systems Engineering - An Introduction, Springer, Heidelberg Berlin NewYork, 1993.
17. P. Naur, B. Randell, and J. N. Buxton, editors. *Software Engineering - Concepts and Techniques*. New York, 1976. NATO Science Committee, Petrocelli/Charter. Proceedings of the Nato Conferences at Garmisch, Oct. 7-11, 1968 and at Rome, Oct. 27-31, 1969.
18. D. L. Parnas, Education for Computing Professionals, in IEEE Computer, Vol. 23, pp. 17-22, January 1990.
19. D. E. Avison, G. Fitzgerald, Information systems practice, education and research, in Journal of Information Systems, Vol. 1, pp. 5-17, 1991.
20. R. S. Pressman, Software Engineering, A Practitioner's Approach, 3rd Edition, McGraw-Hill, New York, 1992.
21. T. W. Olle, J. Hagelstein, I. G. Macdonald, C. Rolland, H. G. Sol, F. J. M. van Assche, A. A. Verrijn-Stuart, Information Systems Methodologies, A Framework for Understanding, 2nd Edition, Addison-Wesley, Wokingham, England, 1991.
22. J. Iivari, A paradigmatic analysis of contemporary schools of IS development, in European Journal of Information Systems, Vol. 1, No. 4, pp. 249-272, 1991.
23. R. Hirschheim, H. K. Klein, Four Paradigms of Information Systems Development, in Communications of the ACM, Vol. 32, No. 10, pp. 1199-1216, 1989.
24. C. Floyd, Outline of a Paradigm Change in Software Engineering. In G. Bjerknes, P. Ehn, and M. Kyng, editors, *Computers and Democracy*, pages 191-210. Avebury, Aldershot, Brookfield, Hongkong, Singapore, Sydney, 1987.
25. N. E. Andersen, F. Kensing, J. Lundin, L. Mathiassen, A. Munk-Madsen, M. Rasbech, P. Soergaard, Professional Systems Development, Experience, Ideas and Action, Prentice Hall, 1990.
26. R. Hirschheim, H. Klein, M. Newman, A social action perspective of information systems development, in J. DeGross, C. Kriebel (eds.), Proceedings of the 8th International Conference on Information Systems, pp. 45-56, 1987.
27. D. G. Wastell, The Social Dynamics of Systems Development: Conflict, Change and Organizational Politics, in S. Easterbrook (ed.), CSVW: Cooperation or Conflict?, Springer Verlag, London Berlin New York, pages 69-91, 1992.
28. G. Bjerknes, Dialectical Reflection in Information Systems Development, in Scandinavian Journal of Information Systems, Vol. 4, pp. 55-77, 1992.
29. M. L. Markus, Power, Politics, and MIS Implementation, in Communications of the ACM. Vol. 26, No. 6, pp. 430-444, 1983.
30. C. R. Franz, D. Robey, An Investigation of User-led System Design: Rational and Political Perspectives, in Communications of the ACM, Vol. 27, No. 12, pp. 1202-1209, December 1984.
31. K. Lyytinen, R. Hirschheim, Information systems failures - a survey and classification of the empirical literature, in Oxford Surveys in Information Technology, Vol. 4, pp. 257-309, Oxford University Press, 1987.
32. R. Hirschheim, M. Newman, Information Systems and User Resistance, Theory and Practice, in The Computer Journal, Vol. 31, No. 5, pp. 398-408, 1988.
33. D. E. Avison, A. T. Wood-Harper, Multiview: An Exploration in Information Systems Development, Blackwell, Oxford, 1990.
34. B. Dahlbom, L. Mathiassen, Computers in Context, Blackwell, 1993.
35. C. Floyd, F.-M. Reisin, G. Schmidt, STEPS to Software Development with Users, in C. Ghezzi, J. A. McDermid (eds.), ESEC'89, Lecture Notes Computer Science, Springer Verlag, pp. 48-64, 1989.
36. P. J. Denning, Beyond Formalism, in American Scientist, Vol. 79, pp. 8-10, 1991.
37. G. Bjerknes, P. Ehn, M. Kyng, Computers and Democracy - A Scandinavian Challenge, Avebury, Aldershot, 1987.
38. P. Ehn, Work-oriented design of computer artifacts, Erlbaum, Hillsdale, 1988.
39. G. Bjerknes, B. Dahlbom, L. Mathiassen, M. Nurminen, J. Stage, K. Thoresen, P. Vendelbo, I. Aaen (eds.), Organizational Competence - A Scandinavian Contribution, Studentlitteratur, Lund, Sweden, 1990.
40. J. Greenbaum, M. Kyng (eds.), Design at Work, Erlbaum, New Jersey, 1991.
41. D. Schuler, A. Namioka, Participatory Design, Principles and Practices, Erlbaum, 1992.