User Participation and Participatory Design: Topics in Computing Education

Karlheinz Kautz
Norwegian Computing Center

ABSTRACT

User participation and participatory design (PD) have not yet been topics of central interest in the context of formal education for computing professionals. This article addresses this subject. 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 informatics. All these approaches have shortcomings as they each relate mainly to a technical-oriented paradigm, which pays little attention to other vital aspects of computing, namely organizational, social, and political ones. Therefore, the curriculum debate is widened and it is argued why user participation and PD (as approaches that deal with these issues) should be part of a comprehensive computing education. An example of an actual course program and the description of one particular course demonstrate how these subjects can be integrated in computer science and system development education.

Karlheinz Kautz is a computer scientist with an interest in organizational impacts of information technology, evolutionary systems development and information system quality; he is a Research Scientist in the Information Technology In Practice group of the Norwegian Computing Center.
1. INTRODUCTION

User participation in the development of computer-based systems has been subject to research and has been exercised for more than 20 years in Scandinavia. Its history goes back to the early 1970s, when a project was initiated in Norway to incorporate the employees’ perspectives in the development and introduction of new computer technology (Nygaard, 1979). User participation relates to the involvement of users in development activities with varying form and degree of involvement and influence (Bjerknes & Bratteteig, 1995). In recent years, user participation in design, or participatory design (PD), as it has been termed, has also gained more and more attention as a topic in the field of computing in North America. The biannual conference on PD (Czyzewski & Johnson, 1990; Muller, Kuhn, & Meskill, 1992; Trigg, Andersen, & Dykstra-Ericson, 1994) and the special issue of the Communications of the ACM (Kuhn & Muller, 1993) underline this trend.

But although there is an intensive curriculum debate going on both in the United States (Denning, 1992; Denning et al., 1989; Hartmanis, 1992; Tucker & Barnes, 1991) and some European countries such as Germany (Bonsiepen & Coy, 1992; Broy, 1993; Müller, 1993; Wendt, 1993) and Norway (Bratteteig, Karabeg, Gran Larsen, & Winther, 1993), these topics are rarely addressed in the context of formal education in universities, colleges, and schools. The discussion is still dominated by traditional mathematical-, engineering-, and system development-based approaches which, apart from all their merits, have some obvious shortcomings. The aim of this article is to widen the curriculum debate and to argue why user participation and PD should be part of a computing curriculum and how these subjects can be integrated in computer science and informatics education.
In Section 2 of this article, the general state of the current curriculum debate, especially in the United States, is presented. This serves as a background for the succeeding, more detailed considerations. In Sections 3 and 4, a closer look is then taken at the reasoning behind mathematical- and engineering-based curricula. In Section 5, traditional system-development education as another basis for current syllabi for software development courses is discussed. The identified deficiencies lead to the argument for teaching user participation and PD issues to computing students in Section 6, where the example from the Department of Informatics at the University of Oslo, Norway, is also used to demonstrate how such a course program could look.

2. THE CURRENT CURRICULUM DEBATE

As American models tend to have a significant influence in the field of computing, the curriculum debate in the United States has been chosen as a starting point for the subsequent reflections. In 1989 an Association of Computing Machines (ACM) Task Force on the Core of Computer Science published its recommendations, which placed an emphasis on software development (Denning et al., 1989). These recommendations are deeply rooted in a traditional perspective of the discipline with mathematics and engineering as its fundamentals. The committee suggested that the name of the discipline be changed to “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. Neither is there a recognition of the consequences of computing for social and working life in the curriculum.

A Joint Curriculum Task Force formed by the ACM and the Institute of Electrical and Electronic Engineers (IEEE) Computer Society published its recommendations in 1991 (Tucker & Barnes, 1991; for more comments see also Denning, 1992, and Hartmanis, 1992). 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. Still, no explicit mention is made of more concrete social and organizational implications of the development and use of computer-based systems and of approaches like user participation
or PD. In the German-speaking countries of Europe the debate goes along similar lines. There, however, we find one explicit proposal for a software ergonomics education that incorporates some of these topics (Maass et al., 1993).

Mathematics and engineering indisputably have been essential in the unfoldment of the computing field. In the next two sections I therefore discuss in more detail how they contribute to educational curricula.

3. MATHEMATICAL-BASED EDUCATION

Dijkstra is a representative of those advocating a mathematical-based education. He contributed to the curriculum discussion in 1989 with his article entitled “On the Cruelty of Really Teaching Computer Science” (Denning, 1989). From his point of view, computers are technical artifacts with a complexity that exceeds all other technical artifacts so far. However, the only things he can see a computer do are manipulate symbols and produce results of such manipulations. A program is an abstract symbol manipulator that can be turned into a concrete symbol manipulator by supplying a computer to it. Dijkstra presented benefits of seeing programs as formulas and arrived at the conclusion that computer science is concerned with the interplay between mechanized 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 admitted 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’ workbench are misleading and give the impression that we are dealing with a known and easy-to-control technology. He warned of an unreflective 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 called the “correctness problem” and the “pleasantness problem.” The former deals with the question 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 (1991) took a similar stand. On the basis of a report of the Computer Science and Technology Board in the United States, he also
argued that a more rigorous use of mathematical techniques, including formal methods and mathematical proofs, would 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 Gries's 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 the dealing with algorithmic concepts.

The pure mathematical approach to software development has been widely criticized. Among others, Winograd (in his answer to Dijkstra's article) argued that computers are not just dealing with mathematical objects (Denning, 1989). 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 mathematical-based argument is seen in the idealized 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 that 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 (Denning, 1989) concluded that it would be foolish to ignore the value of the abstract mathematical skills Dijkstra advocated, 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 that would enable 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.

4. ENGINEERING-BASED EDUCATION

Engineering-based education understands computer science, information system development, and software development as engineering disciplines. Lately, even the term information systems engineering has emerged (see Salvberg & Kung, 1993). The term software engineering was coined 25 years ago on the background of the so-called software crisis, which was manifested by software systems that 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 economical 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 defined *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 ..." (Naur, Randell, &
Buxton, 1976).

It is obvious that the engineering approach aims at technical solutions.
It is interesting to note that the approach was launched in a situation where
the software systems to be developed were thought to be used mainly in
technical contexts (e.g., controlling aircraft or telephone circuits), to sup-
port engineers (e.g., 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 (1990) related computing explicitly to engineering and put for-
ward the following argument. He argues that most computer science
graduates end up working in engineering jobs; thus, they (as such) work as
engineers because 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 development of the early curricula in the 1960s. Mathe-
matical- and engineering-based topics were compressed into quick, shal-
low courses; the biggest part of the curriculum was formed by subjects that
were understood as the "good stuff." This mirrored the then-actual re-
search interests like programming languages and accompanying language
compilers. As a result, Parnas (1990) 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
that emphasizes the classical fundamentals of engineering. Parnas (1990) 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 that 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 prin-
ciples of structured programming, analysis and design of algorithms and
data structures, technical documentation, and systems architecture. Parnas
(1990) and Dijkstra (Denning, 1989) 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, programming techniques, documentation, and quality control procedures, has to be part of the equipment of a system developer. Without the basic technical know-how, no trade can produce quality products. As Winograd (in Denning, 1989) put 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 requested a more practically oriented education that would go 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 being embedded in situations of use.

But within the engineering perspective, emphasis tends to be merely on the technical construction of software. There appears to be a belief that requirements of software components can be described definitely and completely. The engineering perspective seems to suggest that it is these kinds of facts upon which computer professionals can base their work, and that they have nothing to do with work organization, work activities, and requirements analysis.

There is also the opinion that conformity between specification and code is the adequate criterion 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 that, although they disagree about the importance of formal methods, are supported by representatives of the classical mathematical and the traditional engineering schools.

This approach largely excludes the problems arising from determining requirements and proposing functional specifications. This is not that much of a problem in technical areas where requirements are frequently fixed. But system development takes place in a context. Computer and information technology frequently is embedded in organizations; changes of the requirements are often triggered by changes in the organization or have their basis in factors that 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 not only know engineering, but who also can deal with these kinds of problems.
5. TRADITIONAL SYSTEM DEVELOPMENT EDUCATION

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

The International Federation of Information Processing (IFIP) Technical Committee on Information Systems represents those active in this field. By 1968, the IFIP Technical Committee for Education had 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 those whose primary interests are in business and administration and to whom computer technology is only one management tool among others would be in need of education (Avison & Fitzgerald, 1991). The topics of information system development and use attracted business schools. In fact, there still is a clear distinction in the United States between the field of information systems—of which 80% is taught in business schools—and computer science (Avison & Fitzgerald, 1991).

Information systems development and software engineering have a number of elements in common when it comes to software development. Both have to deal with project management and organization as well as with methods and tools for the development of software. They have different starting points: information systems development in system analysis and design, focusing on requirements and functional specifications; software engineering in coding and technical design, stressing 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. For example, Pressmann (1992) used the development of home security systems throughout his whole text. In information systems literature, on the other hand, the examples deal with applications in which people in organizations explicitly are involved. Olle et al. (1991), for example, presented a flight reservation system to demonstrate system development methodologies. Iivari (1991) went as far as to identify 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 draw a distinction between them at all. It looks as if the software engineering community, at least to a certain degree, has recognized the significance of
other than mere technical issues for software development. On the other hand, the information systems community seems to acknowledge the benefits of some engineering discipline.

Then, however, taking seriously that software components are not merely a technical artifact, but parts of organizations, means including issues like establishing requirements, specifying functions, and, in particular, dealing with users and user organizations in a computing studies curriculum independent of a particular perspective.

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

However, in a conventional perspective organizations are seen as unitary structures with a manifest, rational, and, for the most part, hierarchical organizational reality. This reality consists of objects, properties, and processes that are directly observable. Thus, requirements specifications and design descriptions are considered clear-cut documents that are as objective as possible. The resolution of contentious issues within the organization 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 as making system development more rational, placing less reliance on human intuition. Management provides the objectives and dictates the ends for system development projects. System developers take the objectives and turn them into a constructed product. Users operate or interact with information technology to achieve organizational objectives.

If social issues are considered at all in this view, it is 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 view with traditional organizational and management theorists. Management as a discipline of communication has been interpreted, under the influence of Taylorism, 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 and the staff carries them out.

This description is admittedly rather overstated and one-dimensional. Actual practice has moved on from there; however, this type of management is still very often employed. The approach has been used successfully when formalized application areas are concerned. But it leads to some limitations when support for less formalized activities and structures and more dynamic organizations is the aim. Specifications, then, often com-
prise only outdated requirements and do not capture the "real" working practice in an organization. Thus, the information technology applied does not support the work tasks of its users, and the quick and flexible adaptation to frequently changing organizational structures and tasks is hardly possible. The literature is full of stories that report information technology failure.

This is what Hirschheim and Klein (1989) called the orthodox approach to information system development, and it has much in common with what Floyd (1987) called the traditional, product-oriented perspective on software engineering. It takes note of the existence of an organizational context; however, it is very close to the engineering perspective because the general orientation is toward technical issues including some behavioral consequences. But it is not oriented toward social issues as a main subject of concern. Thus, including organizational issues in a curriculum for computing professionals is a step in the right direction, but it is not sufficient.

6. TEACHING USER PARTICIPATION AND PD

System development, according to Andersen et al. (1990), consists of all those activities that aim at changing an organization through the use of computer technology. Software development as part of system development is not just a process of technical change, but first and foremost, it means organizational development. Hirschheim, Klein, and Newman (1987) argued that system development that aims at the use of computer technology is a social process that relies on technology.

Wastell (1992) and Bjerknes (1992) described this social process as a dialectical reality of mutual reciprocating influences and contradictions in which organizations are seen as pluralistic structures. Many points of view and interests exist there. This makes system development a complex, social phenomenon. The political dimension, for example, is stressed by Markus (1983) and Franz and Robey (1984). Solutions and prescriptions that oversimplify the actual reality will hardly lead to success.

System developers must try to understand the organizational, social, and political context of workplaces. In traditional approaches, this context is abstracted away to pure information processing aspects. There, system developers use structured analysis and design methods that allow them only to analyze and describe the technical features of organizations (cf. Markus, 1983). The failure of numerous system development endeavors due to neglecting these aspects is frequently reported (see, e.g., Hirschheim & Newman, 1988; Lyytinen & Hirschheim, 1987).

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 organization when information technology is to be introduced. The approaches may be distinguished by the different roles the system developers may play in development projects. Avision and Wood-Harper (1990) made a distinction between system developers as facilitators, as agents for social progress, and as emancipators. Similar distinctions can be found in Hirschheim and Klein (1989) and Dahlbom and Mathiassen (1993). 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 center of interest away from organizations as abstract structures. The center of interest is moved towards the everyday working practice of people within organizations. 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. Already in 1977, Bjørn-Andersen and Hedberg (1977) postulated three reasons why users might be actively involved in the design of computer-based systems:

1. An improvement of the knowledge on which the construction of the systems is based.
2. Reduction of resistance and development of realistic expectations.
3. Increase of workplace democracy.

These reasons are rather different; the first two are more practical, whereas the third is more politically biased. However, the general need for user involvement is acknowledged. In line with Floyd (1987), Denning (1991) saw also a new paradigm for software development here (see also Floyd, Reisin, & Schmidt, 1989). This paradigm has become known by the terms user participation or PD. Denning considered 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 argued that, in this respect, it is superior to traditional approaches to software development.

The underlying concepts and examples for PD are described, for example, in Bjerknes, Ehn, and Kyng (1987), Bjerknes et al. (1990), Ehn (1988), and Schuler and Namioka (1992). Techniques like future workshops, mock-up construction, and cooperative prototyping are described in Greenbaum and Kyng (1991). The material is well-documented and can be taught. At the University of Oslo, the System Development group at the Department of Informatics provides three different courses on different graduation levels, which bring a PD philosophy generally into the computing education of the department. All three courses are organized as a combination of lectures, small practical exercises, and more extensive project work. They are organized as a track starting with Computers and
Society at the introductory level, continuing with System Description and Language at the advanced level and finishing with System Development—Theories and Models at the graduate level. The courses are not compulsory for all students; only those specializing in system development have to sign up. Nevertheless 10% to 15% of all students enter the program. The courses are briefly presented in Sections 6.1 to 6.3.

6.1. Computers and Society

This course is held at the introductory level. It aims at giving computer science students the background for understanding the consequences of the introduction and use of computer-based systems in a larger social context. Students learn that there exist different interest groups in system development projects. The course provides the qualification to identify and analyze problems and disagreements with regard to the utilization of information technology. A special emphasis is put on introducing the laws and directions that regulate the work of computing professionals in Norway.

In the lectures, representatives from research institutes, government authorities, trade unions, and other social groups put forward their perspectives and opinions about computer science. The exercises and the project work are performed in small working groups to contribute to students' abilities to work constructively in teams and to resolve emerging conflicts. In the projects, students explore a particular topic of the lectures in more detail. Recently, students investigated the use of Internet services in organizations. By interviewing the different stakeholders in organizations such as a daily newspaper and a library, students are confronted with organizational reality; although not involved in actual PD projects, they get a realistic feeling of vital aspects of technology use.

6.2. System Description and Language

This course is held at the lower graduate level. It aims at giving the students an understanding of the relation between descriptions and reality and points out the difference between natural and artificial languages. It also discusses formal and semiformal system descriptions with regard to other analysis and design techniques used in system development. Because one emphasis is on the practical use of description languages, two different languages are taught extensively. This enables students to characterize means of representation from different perspectives and allows them to judge their benefits and shortcomings. Thus, a basis is provided for selecting an appropriate description instrument for a given situation.

The small exercises are construction-oriented. Students prepare for the project by learning techniques like object-oriented analysis and design and
the use of rich pictures from Soft Systems Methodology (see, e.g., Checkland, 1981) or prototyping (Budde, Kautz, Kuhlenkamp, & Züllighoven, 1992). In the project itself, two tasks have to be solved in two 4-week periods. The task of the 1995 spring term consisted of the analysis of the use of electronic mail systems in a couple of organizations. The analysis had to be performed by using the previously learned description techniques, and the resulting report had to be approved by the interview partners.

The report is then used in the second part of the task as a kind of specification for a possible mail system. The students have to decide which part of the descriptions to realize as an executable prototype. The construction and test of the software concludes the projects. Again, although no users participate in the actual design work, the students learn to pay attention to user perspectives and get hands-on experience with techniques that have their basis in PD.

6.3. System Development—Theories and Models

This graduate-level course tries to put the contents of the earlier courses into a common perspective. It aims at providing the means to organize and conduct larger system development projects and processes. Different models for performing projects (e.g., the waterfall model) and evolutionary approaches are introduced and contrasted with each other. Literature studies are used to expound upon problems of the relation between organizational and system development. The challenges of planning and steering projects and different ways of handling these are topics of the lectures. The problems inherent in the cooperation of developers and users and various approaches of mastering them are another main point of the course.

The accompanying project work is used to illustrate the theoretically introduced issues. This year's assignment was related to a project that developed a technical infrastructure for a large Norwegian public sector organization. A critical analysis of the project organization and its implications on the project results was the students' task. Key participants and representatives of all the groups affected were to be interviewed and a report was to be delivered.

This course focuses on the development and use of computer-based systems from an organizational perspective. It is supplemented by a varying number of nonregular courses and seminars that are more closely related to the design and construction activities of system components. One such course is concerned with the design of user interfaces. None of these courses, however, include direct and acting participation of users in design activities, but they demonstrate important aspects of user involve-
ment. I have conducted another of these courses, entitled Prototyping—An Evolutionary Approach to System Development. Previously, it has been conducted successfully at the Technical University of Berlin. It is explained in more detail in Section 6.4.

6.4. Prototyping: An Evolutionary Approach to System Development

The aim of this course is to pass on basic concepts and different techniques of prototyping as a strategy for PD and to underline the active role of the users in the design process. In the lecture part of the course, the concepts of prototyping (Budde et al., 1992; Floyd, 1984) are presented in depth. In the practical part of the classes, prototyping projects are performed by graduate student groups. The technique chosen is very similar to what is described by Grønbæk (1991) as cooperative prototyping. The groups’ task was to develop prototypes for an interactive information system to support the management of bibliographical data and literature reference lists. To give the students a feeling for the problems encountered in PD, the projects were run as a kind of role play, where different groups with different tasks were represented. Each group has six to nine members: Two or three members adopted the users’ perspective, two to four members acted as developers, and two members served as prototyping and method consultants.

Each group was provided with an identical three-page document informally describing basic requirements for the target system. The groups could choose between two traditional description languages and five prototyping tools, ranging from stand-alone screen generators to Computer Aided Software Engineering tools and HyperCard™. As a means for organizing and reflecting the project work, project diaries were introduced. The time span of the projects was restricted to 8 weeks. Weekly project meetings and a written project report on the project’s development, as well as the usual product documents accompanying the prototypes, were required as part of the final result. More comprehensive discussions of various aspects of the course can be found in Gryczan and Kautz (1990), Kautz (1992), and Kautz (1993a).

The role play approach is fruitful. Students take the assigned roles very seriously. During the projects, they usually go through a couple of sincere conflicts emerging between those representing the users and those acting as developers. This gives them a realistic foretaste of their future work, and students report that they experience potential problems and solutions of PD efforts very explicitly (Kautz, 1993b).

These examples show that it is feasible to teach issues of user participation and PD both in theoretical and practical ways. Together with all the other topics related to system development, they offer an education for computing
professionals that does not omit important parts, but that comprises all vital aspects of system development. This kind of system development education should be firmly established in any computing curriculum.

7. CONCLUSION

Different approaches to computing education have been presented in this article. The mathematical, engineering, and system development approaches supplement each other, and a curriculum must comprise all of them. It is not sufficient, however, to stick to the traditional, merely technocentric approaches. A contemporary education also has to include approaches like user participation and PD. This is necessary, because computing professionals have to be prepared to meet not only technical, but also organizational, social, and political challenges, and it is possible, because the existing knowledge is fairly well-documented. An example of the successful implementation of PD issues in a course program has been given in this article. Further discussions should address how these topics can be more widely integrated in computing curricula. For this purpose, it will be useful to collect and compare more existing teaching experience.

NOTES

Background. An earlier version of this article was presented at the 1994 Participatory Design Conference in Chapel Hill, North Carolina.

Acknowledgments. Numerous people supported the work with this article. Thanks are due to Barbara Andrews, Tone Bratteteig, Joan Greenbaum, David Hakken, Michael Muller, and Terry Winograd, who were the members of a panel and organizers of a workshop which was dedicated to the topic education and participatory design at the PDC’94 conference. Randy Trigg and Susan Anderson encouraged further work with the manuscript after the conference. Heinz Züllighoven and Guido Gryzcan taught the first prototyping course with me in Berlin. Annita Fjuk, Pål Sørgaard and Kristin Braa contributed the background information about the courses in Oslo, and the four anonymous reviewers provided valuable comments on the earlier version of the paper.

Author’s Present Address. Karlheinz Kautz, Norwegian Computing Center, P.O. Box 114 Blindern, N-0314 Oslo, Norway. E-mail: Karl.Kautz@nr.no.

HCI Editorial Record. First manuscript received January 15, 1995. Revision received October 1995. Accepted by Randall Trigg and Susan Irwin Anderson – Editor.

REFERENCES


education as part of computer science studies at German universities]. *Informatik Spektrum*, 16(1), 25–30.


