Education and Information Technologies 5:2 (2000): 103–117

DEVELOPING EDUCATIONAL SOFTWARE: A PROFESSIONAL TOOL PERSPECTIVE

ENRIQUE HINOSTROZA AND LUCIO E. REHBEIN
Instituto de' Informática Educativa, Universidad de La Frontera, P.O. Box 380, Temuco, Chile
E-mail: ehinost@iie.ufro.cl ; lrehbein@iie.ufro.cl

HARVEY MELLAR AND CHRISTINA PRESTON
Science and Technology, Institute of Education, University of London, 20 Bedford Way, London, WC1H OAL, UK
Email: hgm@ioe.ac.uk ; temsccp@ioe.ac.uk

The selection, and use of educational software and its impact in schools are still controversial issues. In this paper we present an alternative conceptualisation of educational software based on considering the software as an instrument for teachers' professional performance. We review previous work in the areas of the design, development and evaluation of educational software and of the process of educational innovation. The review of these four areas converges to demonstrate the need for knowing and considering the context of use of educational software and for understanding users' perspectives about its roles and possibilities and hence supports a perspective on educational software which sees it as a professional tool for teachers' performance of their teaching role.

Keywords: educational software, pedagogy, evaluation, professional tool, cognitive tool.

Introduction

The use of educational software in schools is still an arena for debate and controversy. There is the software development industry with its growing market of new multimedia products (Broderbund, Microsoft Corp., TAG Development, The Learning Company, Tom Snyder Productions, Unlimited, ZETA Multimedia, etc.)[1]. On the consumers' side, there is evidence that the role of information technology in schools is controversial (Lowther and Sullivan, 1994) or at least that its effects are not conclusive (Johnson et al., 1994), and that software products that are most frequently used in school are based on drill and practice activities (Evans-Andris 1995; Cuban 1997). Then there are research groups producing a growing number of reports focusing on the teaching and learning processes using particular pieces of software (diSessa et al., 1995; Laborde 1995; Mellar et al., 1994; Schwartz et al., 1993; Soloway and Pryor 1996).

These three groups use and offer different software products and have very different views as to how to assess their value.

In this paper we analyse this apparent dissociation, presenting evidence and ideas that might help to identify the root of the problem. The life cycle of a piece of software is reviewed, that is, from its design and development, up to its use, evaluation and insertion in schools as an educational resource.

We wish to argue in favour of the need for research into the concept of educational software from a situated perspective, and in particular for research into teachers' understanding of the role of educational software and their concepts about it. This argument leads us to wish to emphasise a perspective on educational software which sees it as a professional tool for the teacher's performance of their teaching role.

Educational software design

In this review we will analyse the design of educational software from the perspective of the intentions of the author, that is from the underlying teaching and learning principles that can be found in the software. Accepting the difficulty of knowing the 'real' intentions of the author, we will focus on the explicit elements of the design and because these elements have been largely captured by the different classification methods proposed in the literature, we start by looking at educational software classifications.

'Learning with Software' (Open Learning Technology Corporation 1995) presents an overview of these systems of software classification. We simply present here the organisation given in that paper with some additional references of our own (text directly quoted from the web site is indicated by quotation marks).

- **By subject:** "Essentially this system provides for classification of software by school subjects. For example, all those software programs that can be applied to, say, Social Studies."
- **By type** (i.e. the functionality built in to the software): The categories defined under this type of classification are, for example. Tutor, Tool and Tutee (Taylor 1980). Another approach to classification of software fitting into this category is given by Chandler (1984) using the labels: Tutorial, Games, Simulation games, Experimental simulation, Content free tools, and Programming languages.
- **By educational paradigm** (i.e. the learning paradigm that is embedded in the software): One such classification comprises four paradigms: Instructional, Revelatory, Conjectural, and Emancipatory (Kemmis et al., 1977). Another classification in this group is that proposed by Laurillard (1993). Starting from a definition of the learning process, she defines the categories Discursive, Adaptive, Interactive and Reflexive.
- **By use** (i.e. the teaching strategy that could be triggered by the software or is embedded in the software design): "Fatouros et al., (1994) offer a classification of computer aided learning based on the key domains or learning areas that teachers plan for young children to explore (i.e. images, sounds, text, stories and ideas, facts and figures, consequence)." This kind of classification was also used by Watson (1993), who focused on the 'Educational Activity' that is supported by the software. He proposes the categories Information Gathering, Analysis and Evaluation, and Presentation.
- **By impulses to learn:** This category is based on a taxonomy proposed by Bruce (1996), who uses the ways in which educational resources support integrated, inquiry-based learning as a classification key. He defines four broad categories: enquiry, communication, construction, expression

Each of these classifications serves a particular purpose of analysis and comparison. For example if the aim is to build a library of software to be consulted by teachers, the subject classification could be used; if the aim is to compare the effects of software on students' performance, then the educational paradigm could be used. But in examining these classifications, we can see that none of them is based on the implications of the design for what *a teacher* would do with the software in the classroom.

Educational software development

In order to analyse the process of software development, we present a brief look at the general issues of software engineering and the methods proposed in this area, and then present some reported examples of the software development process.

Software Engineering

The concept of the software life cycle (Sommerville 1989) structures the stages of software development. Based on this model there have been several propositions, including the waterfall model and the revised waterfall model. Other models propose a sequence of stages that allow the developers to reduce the risk involved in the development by introducing reiterative tests of a prototype, which evolves within the project. Prototypes allow developers to acquire information about the requirements, technical feasibility, and other risk elements, but this method isn't well integrated with the end product because of its partial and ill integrated step by step development. Boehm (1988) proposed a new approach to this development method, which has been named the spiral model. It integrates the waterfall model with the evolution of prototypes.

Despite the variety of different methods or techniques used for software development, it is still difficult to produce a piece of software. Winograd (1995) argues that one of the key differences between software and most other kinds of artifacts that people design is the freedom of the designer to produce a world of objects, properties, and actions that exist entirely within the created domain. This special condition of software products makes it very difficult to share the idea of the product to be developed, because the artifact belongs to the author and it is not possible to apprehend it in the computer, merely to realise it through every attempt to use it.

Misunderstandings between software designers and software users are well known. Colin Potts (1993) in his article 'Software Engineering Research Revisited' suggests that an emerging trend exists today towards using industry as the laboratory for discovering new techniques of developing software, and that this tendency emphasises the relevance of an empirical definition of the problems, the study of actual cases and their contextual aspects. Hughes et al., (1995) in arguing for a role for ethnography in software development, claim that it is vital for designers to understand the work setting as a socially organised entity prior to initiating the design stage.

Development of educational software

There are two main approaches taken by different educational software design teams to explicitly, or implicitly, tackle the problem of context. The first is transferring the software design responsibility to the teacher, and the second is to incorporate teachers as part of the design team.

The first approach means that teachers, by themselves, design a piece of software that they find to be useful for their activities (cf. Fitzgerald et al., 1992). This has been tried in different empirical situations but the essential problem here is that the highly specialised technical knowledge required (software engineering, programming methods and techniques, human computer interface design, etc.) to produce a professional piece of software is beyond the normal training of teachers (and other professionals as well). Software produced in this way can be very useful in bounded situations, but is certainly not satisfactory in general, either for external evaluators or for teachers themselves (Hoyles et al., 1991).

The second approach entails the involvement of one or more teachers in the software development process with defined roles and activities. In Char and Hawkins' (1986) report teachers were part of the design process, advisors on curriculum issues and evaluators in different stages of the software development process. Hawkins and Kurland (1986) describe an example of requirement specification for the design of information-managing tools in the schools. In both projects, although there was a prior conception of the piece of software to be developed, there was no stage of 'requirements specification' in order to analyse what was really needed in the school(s).

There are a few good examples of approaches to the use of technology in education based on requirements analysis, one in particular is McConnell's course design (McConnell 1994), which starts with an analysis of co-operation and learning. From this he designs strategies for using the technology to support the different ways of implementing computer supported collaborative learning.

However, there seems to be little experience of development of software based on an analysis of the needs and activities found in the schools or in the classrooms. Almost all the examples of software development reviewed were based on a preconceived idea of what kind of software should be developed and teachers and students were incorporated into the process after this initial definition or conception.

Software evaluation

For the purposes of this paper we will understand software evaluation as a formal procedure that helps someone else to build up a judgement about the software, in the sense of its effectiveness in the areas or activities for which it has been designed or is been used. For the moment we will exclude the process of selection and review of software described by Squires and McDougall (1994) and Winship (1989). We classify the different evaluation techniques into three groups:

- **Experimental methods:** In this approach the experimental method of pre and post tests using experimental and control groups is used in order to assess the effectiveness of a piece of software. Examples of these methods are found in reports by Reiser and Dick (1990) and Zahner et al., (1992), in which they propose a specific method of performing an experiment to evaluate software.
- **Check-list approach:** This group of methods is based on applying a set of predetermined criteria to a piece of software. Examples of such methods are found in Tolhurst's (1992) study, and Squires and McDougall (1994) provide an extensive review of the different options.
- **Qualitative evaluation:** Some evaluators argue for the evaluation of the software in a situated context and propose the application of qualitative methods to evaluate software. Examples of this are given by Crook (1991), who changes the focus of analysis of software from the interaction with the computer to the interaction around the computer, and Squires and McDougall (1994), who propose a method of analysis of software based on three interaction paradigms: teacher-student, teacher-designer and student-designer.

One of the main problems of software evaluation is the nature of software itself, in that it is not readily accessible. Unlike a book or other teaching material it is not accessible to inspection, by skimming, scanning, browsing, etc. It must be run and explored on the computer and the teacher will need a certain expertise to be able to do this (Squires and McDougall 1994). Winograd (1995) describes the particularity of software design as the creation of an independent artifact that exists and has sense in a defined domain. In a similar vein Olson (1988) defines software as 'ideaware'. Crook (1991 and 1994) concentrates on the concept of interaction between users (students and teacher), defining a social context in which this interaction occurs.

So, in order to evaluate a piece of software, we argue that it is not possible to assess it as an isolated element, rather it needs to be evaluated in a specific social context and set of circumstances.

Educational innovation

It is commonly argued that computers and telecommunications are key tools that permit (and eventually produce) the change (innovation) from the traditional bureaucratic culture of organisations to a new professional culture. Moreover, in many cases the justification for investing in information technology is based on the need to innovate in several areas or dimensions, rather than in the need for the technology itself. In fact, speaking about the rationale for innovation in schools, Grunberg and Summers (1992), rephrasing Fullan (1982), say: "schools tend voluntarily to adopt innovations which promote their image as up-to-date and efficient" (p. 259). Therefore, besides efficiency, effectiveness, and other considerations, computers are often seen as innovation 'symbols' or 'signs' and once introduced, act as catalysts in the process of change (Hawkins et al., 1990).

Computers produce a wide range of effects in an organisation and much has been written about the different levels on which they impact. One interesting specific effect of computers in schools is reported by Olson (1988), who identifies two different ways in which teachers use the computer:

- Expression tool: The computer is an instrument to express how they want themselves to be seen as teachers.
- Trojan horse: The computer is used as an aid to innovate in the teaching strategy.

These ways of using the computer were not perceived by the teachers themselves, but were revealed by the process of analysis and interpretation that the researcher applied to his observations of their work with computers. In both cases it is possible to ask whether this is a computer driven or a computer supported process. In the former, technology plays the role of catalyst, and in the latter it is a support for the ongoing process of change.

This view of the role of technology is coherent with that of Winograd and Flores (1986), who argue that when a change is made, the most significant innovation is the modification of the conversation structure, not the mechanical means by which the conversation is carried out. This focuses the ideas of change not on the technology but on the relation or activities that are carried out with the technology, in this case, teaching and learning.

Placing technology in a supportive role implies first, understanding the on-going process of change and second, choosing an appropriate technology. This view is coherent with the reflexive concept of change described by Olson (1988), who defines teacher behaviour as reflecting on action, research activity as understanding teacher intentions and innovation activity as engaging in critical analysis of practice.

These ideas are in-fitting with the overall picture presented in previous sections, they shift the focus of innovation away from the technology and closer to the context and actions, that is to a situated perspective on change.

Three conceptions of educational software

The previous sections have demonstrated the need for considering a more specific perspective of software design, development, evaluation and introduction. Starting from these arguments, we present some alternative conceptions of educational software.

Cognitive tools: learning centred software design

In this perspective software design and development is viewed as an activity whose goal is to produce a tool that is expected to have an effect at a cognitive level. The design may well be grounded in some learning theory which gives the framework for the software design (the requirements). These theories include behaviourism, constructivism, and others, all of which have a clear element of self-determined learning and therefore share common assumptions in their design. Some software of this kind does incorporate

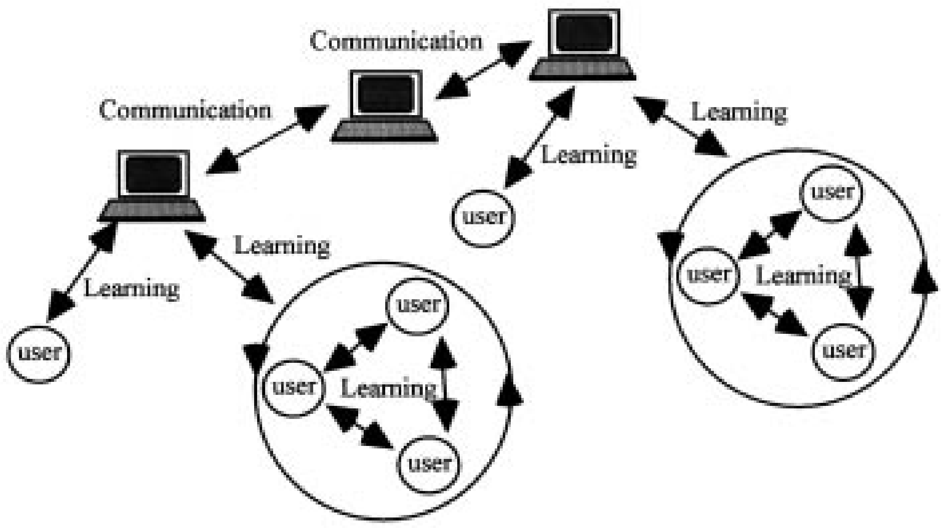

Figure 1. The use of learning centered software

elements of a teaching strategy, but these are embedded in a learning framework which is explicit in the software. From the designer's or developer's perspective, the arena of learning is in the interaction of the student with the computer. Software of this kind can be represented by Figure 1.

Reusser (1993) has software of this type in mind when he describes educational software:

> "computer environments should be seen as mind-extending or catalysing tools for intelligent and volitional learners and virtually autonomous problem solvers. They should provide stimulating and facilitating structures in order to promote meaning construction activities, such as planning, representation and reflection" (p. 146)

Another example of this sort of design is given by Laurillard (1990), who describes two models for teaching and learning: the didactic model and communication model. In the former she speaks about a 'preceptual knowledge' that is transmitted by the teacher to the student. In the latter she describes knowledge as a 'negotiable commodity' between teacher and pupil. Based on the latter model she specifies the following software requirements:

- the student should have direct access to the object domain
- the software should have operational knowledge of the domain
- the software should be able to give intrinsic feedback
- the software should make the goals of the exercise explicit.

As in the previous definition, these specifications exclude the teacher from the learning process, which is not actually the case for all the software included in this group.

Although it is clear that the majority of the actual educational software available today could be classified in this category, from our perspective a possible disadvantage of this kind of software is that the assumptions made during the design may be well grounded in learning theories, but there is an assumption that the computer will be used in a specific way. This assumption demands that the teachers act in a particular way in order to create the situation in which the student can interact with the software in the manner intended by the designer in order that learning may take place.

One might argue that this type of educational software should be evaluated considering learning gains only and without necessarily including the context in which it would be used, in that the context and strategy are determined by the software. This would mean that we would then classify the interactions around the computer (Crook 1994) as side effects, in that they were not intentionally designed.

From an innovation perspective, this type of software could be seen as a Trojan horse, carrying new learning methods and thereby challenging the classroom routines. The underlying assumption is that teachers will use the software instead of continuing with what they are currently doing, because the learning theories embedded in the software are claimed to be better.

Professional tools for teaching: teaching centred software design

This perspective on software design and development has its origins in particular teaching methods, that is it has been conceived as an organisational aid for the teacher in the classroom. The essential difference between this conception of educational software and the previous one, is that the software design has explicit assumptions about how to use the computer in the classroom.

This alternative way of designing educational software integrates the computer into a certain teaching strategy, giving the teacher a special role in the activities. This role is made explicit in the design of the software and is based on a study of teachers' software requirements to help their teaching. The locus of learning is in the classroom activity, not in the student's interaction with the computer, in fact the software may even be designed in such a way that the student does not need to use the software. Software in this group can be seen in Figure 2.

Fraser et al., (1991) describe the different classroom roles that the teacher, pupil, or computer adopt when software is used in a classroom situation. The different roles described are (p. 212):

- Manager (tactical), corrector, marker, computer operator.
- Task Setter, questioner, example setter, strategy setter.
- Explainer, demonstrator, scene setter, image builder, focuser, imitator, rule giver, coach.
- Counsellor, adviser, helper, devil's advocate, encourager, stimulator, listener/supporter, observer, receiver, diagnostician, problem solver.
- Fellow pupil, rule applier, hypothesizer, problem solver.
- Resource, system to explore, giver of information.

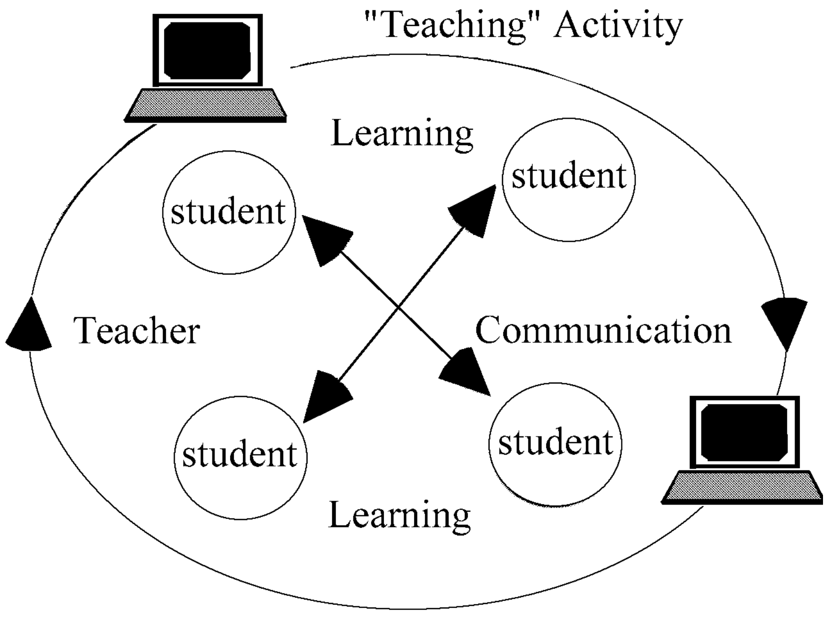

Figure 2. The use of teaching centered software

These roles reflect the behaviour of the teacher and/or students while using a piece of software in a classroom lesson, and they could form a useful starting point for thinking about teaching centered software design. Another example of this kind of approach is reported by Dockterman's (1991) description of producing software to be used in a one-computer classroom situation.

Mercer (1993), describing the implications of the context in learning, writes:

> "1 It implies that the process of learning about, or through, computers is not primarily to do with the relationship between a learner/user and the machine–the 'interface'–or even the software being used. It is instead very much to do with the contextual framework within which the learner/user is doing things with the computer....
>
> 2 It implies that what is learnt by particular children through the use of computers may only be understandable in terms of the history of the teaching-and-learning relationship in which that learning took place...."
> (pp. 31–32)

With this definition, Mercer changes the focus of the software design from the student-machine interaction to the context of use. But he still continues to foreground the interaction between the student and the computer, which is not necessarily required in our description of teaching centered software design.

Further support for the idea that much educational software has a role in the whole classroom situation rather than in the individual-computer interaction, can be found in the work of Olson (1988), who identifies a role for the computer, other than being directly related to a specific teaching activity, in supporting the professional performance of the teacher.

These findings show different roles of the computer as a teaching resource, in the sense that it provides teachers with an aid in performing their job. In this case the aid is not directly related to the specific teaching activity, but to the professional performance of the teacher.

This notion of educational software emphasises the teacher-designer perspective of software evaluation (Squires and McDougall 1994), whilst focusing the object of the software design on what the teacher is able to do with it. In other words, the software would be judged to be successful if teachers could use it as an aid to do better the tasks that they already perform, rather than imposing changes to existing practices. The change or innovation process would be decided by the teacher and supported by the software.

This conception of educational software design suggests that designers should review work on the teachers' professional knowledge (Eraut 1994, Grossman 1995) and their expertise (Berliner 1995, Marton 1994) in order to understand better what teachers need to perform their job (professional requirements), and also to review classroom management issues (Jones 1996) to understand other factors (from the environment) influencing its use (Sandholtz et al., 1997). These areas could throw light on alternative designs of educational software that could fit into teachers' practices and respond better to their requirements.

Teaching material/resources provider

In this group are those packages that serve as a resource to carry out a specific task. The software here does not include an explicit learning or teaching strategy, but it may help to perform learning and/or teaching processes.

The focus of learning could be on the student-software interaction or on the activity organised by the teacher. The computer is conceptualised as a special tool for performing an activity or as a powerful book-like resource.

There are many studies that report the use in schools of 'traditional' software of this type (word processors, data bases, spreadsheets, encyclopaedias, etc.). This software has generally been designed to be used in other environments (industry, administration, library, home, etc.), therefore its introduction into classroom settings is unlikely to emphasise cognitive and pedagogical aspects (Squires 1996). Teachers who use this kind of software often advocate 'vocational' arguments, arguing that students need to be prepared to use this kind of software when they enter into the job market (Squires 1996).

Discussion

Leaving aside the use of software as a teaching material/resources provider, it was argued that educational software design follows one of two approaches:

- Designing cognitive tools: here the authors are trying to build software that implements some learning, cognitive or instructional theory. In doing so they give the computer a high degree of responsibility for the learning outcomes.
- Designing professional tools for teaching: here the authors are trying to find out ways in which a computer could be used as part of the teaching process. They include a more systemic view of the process of teaching and learning, rather than a particular conception of learning with or around the computer.

Despite all the efforts of designers using the first approach, and all the designer's intentions, research shows that the software most frequently used in school is based on drill and practice activities (Evans-Andris 1995, Cuban 1997). In turning to look at the second approach, we are back to the key question of defining the role of the computer in teaching.

Despite high expectations about the use of computers in education, research has shown that in this field the role of the technology remains controversial (Lowther and Sullivan 1994) and its effects are inconclusive (Johnson et al., 1994). In trying to explain this situation there are two main arguments, the first is that the teacher should be more technology literate in order to master the technology, and the second refers to the lack of understanding of the software designers about the teaching/learning process.

In the first group, some authors complain about the capacity of the teachers to understand and/or adapt pieces of software to their classroom activities. Handler (1993) and Winship (1989) complain about a lack of appropriate training and support for teachers who want to use this technology.

In discussing educational software, Winship (1989) comments that:

- teachers find it very difficult to identify software that they believe will be useful in their own teaching.
- much of the existing software is difficult to integrate into teaching because it is either too easy, too hard or it takes too long before useful results are produced.
- often the teachers must put in a great deal of preparation time before the software can be used in the classroom.

There seems to be a discrepancy between what is being offered today as good educational software, what teachers really do with software in the schools and what teachers' expectations are of what could be done with it. In a teacher's words:

> "Given that I am expected to maintain order and get students to learn essential skills, knowledge, and values, how will these machines help or hinder my mission?"
> (Cuban 1997 p. xii)

So, one general critique of the design and implementation of educational software to date is that there is a lack of understanding of what is happening in the classroom and of the discourse of the teachers and their circumstances. Mercer and Scrimshaw (1993) and Olson (1988) argue that we know too little about computer activities in the classroom. Crook (1991) and Koedinger & Anderson (1993) argue that we should understand the discourse of teaching and the instructional context. Winograd and Flores (1986) talk about the general issue of understanding the domain of action of the user, in this case the teacher and the pupil. Reusser (1993) speaks about the pedagogical and didactic philosophy that a software design should incorporate and the importance of the learning and teaching activities that take place in the 'behavioural setting' of schooling.

School software should be designed to be used in the school, for purposes and needs that are present in the school. The implication of this definition is that to design a piece of school software, first it is necessary to know the needs of the school and, from this starting point, to design a piece of software to help to satisfy those needs. It implies not imposing preconceived software designs in order to improve teachers' professional activities, but basing designs upon the actual practices of the teachers, enabling them to perform those tasks more effectively.

We have presented four themes:

- Software design–where we concluded that designing software should incorporate elements from the reality in which it will be used. At present there is a lack of understanding of the role of the teachers and about the activities that occur with and around the software that is being used in the schools.
- Software development–where we presented arguments for a modification of traditional software development methods to incorporate techniques such as ethnography into the early stages of the process. This is in order to understand the professional activity of the software user (the teacher) and use this understanding to design appropriate software.
- Software evaluation–where we presented arguments for the incorporation of the contexts of use as a new dimension for evaluation of educational software, transforming it into a process that could be understood as qualitative research. This highlights the situated nature of software use.
- Educational innovation–which we presented as a phenomenon that is highly correlated with the introduction of IT into schools. In this sense, the role of computers as a support tool for such a process rather than as a catalyst in it was emphasised. Therefore, and in order to be able to play this role, it is essential to understand the contextual setting in which it will be used.

These themes all point to the importance of knowing and considering the reality of use in order to design, develop and evaluate educational software that could be used to support an innovation process which engages the teacher and the school, that is, as professional tools for teachers.

Acknowledgements

This work has been carried out thanks to the support of Fondecyt, project number 1960854; the British Council; the H. E. L. project 'Advanced Computer-based Technologies for Teaching and Learning (ACOTEL)' and the Chilean Ministry of Education, through the MECE programme's Enlaces Project.

Notes

1 All names of companies are respective Trade Marks

References

Anderson, A. et al. (1993) Software style and interaction around the microcomputer. *Computers and Education*, **20**, pp. 235–250.

Berliner, D. C. (1995) Teacher expertise. In Anderson, L.W. (ed.) *International Encyclopedia of Teaching and Teacher Education*, pp. 46–52. Oxford: Pergamon.

Boehm, B. W. (1998) A spiral model of software development and enhancement. *IEEE Computer*, May 1998, pp. 61–72.

Bruce, B. C. (1996) *Educational Technology: Tools for Inquiry, Communication, Construction, and Expression*. On-line at www:http://www.ed.uiuc.edu/facstaff/chip/taxonomy/.

Chandler, D. (1984) *Young Learners and the Microcomputer*. Milton Keynes: Open University.

Char, C. and Hawkins, J. (1986) Charting the course: involving teachers in the formative research and design of the voyage of the mimi. In Pea, R. D. and Sheingold, K. (eds.) *Mirrors of Mind: Patterns of Experience in Educational Computing*, pp. 211–241. Norwood: Abelex Pub. Co.

Crook, C. (1994) *Computers and the Collaborative Experience of Learning*. London: Routledge.

Crook, C. (1991) Computers in the zone of proximal development: implications for evaluation. *Computers in Education*, 17, pp. 81–91.

Cuban, L. (1997) Foreword. In Sandholtz, H.J. Ringstaff, C. and Dwyer, D.C. (eds.) *Teaching with Technology: Creating Student Centered Classrooms*. New York: Teachers College Press.

diSessa, A. A. Hoyles, C. and Noss, R. (eds.) (1995) *Computers and Exploratory Learning*. NATO ASI Series F Subseries Advanced Educational Technology. London: Springer.

Dockterman, D.A. (ed.) (1991) *Great Teaching in the One Computer Classroom*. Tom Snyder Productions.

Eraut, M. (1994) *Developing Professional Knowledge and Competence*. London: The Falmer Press.

Evans-Andris, M. (1995) An examination of computing styles among teachers in elementary schools. *Educational Technology Research and Development*, **43**, pp. 15–31.

Fatouros, C. Downes, T. and Blackwell, S. (1994) *In Control: Young Children Learning with Computers*. Wentworth Falls: Social Science Press.

Fitzgerald, G. E. Bauder, D. K. and Werner, J. G. (1992) Authoring CAI lessons: teachers as developers. *Teaching Exceptional Children*, Winter, pp. 15–21.

Fraser, R. et al. (1991) Learning activities and classroom roles with and without the microcomputer. In Boyd-Barret, O. and Scanlon, E. (eds.), *Computers and Learning*. Wokingham: Addison Wesley & Open University.

Fullan, M. (1982) *The Meaning of Educational Change*. Columbia University: Teachers College Press.

Grossman, P. L. (1995) Teachers' knowledge. In Anderson, L.W. (ed.) *International Encyclopedia of Teaching and Teacher Education*, pp. 20–24. Oxford: Pergamon.

Grunberg, J. and Summers, M. (1992) Computer innovation in schools: a review of selected research literature. *Journal of Information Technology for Teacher Education*, **1**, pp. 255–276.

Handler, M. (1993) Preparing new teachers to use computer technology: perceptions and suggestions for teacher educators. *Computers and Education*, **20**, pp. 147–156.

Hawkins, J. and Kurland M. D. (1986) Informing the design of software through context-based research. In Pea, R. D. and Sheingold, K. (eds.) *Mirrors of Mind: Patterns of Experience in Educational Computing*, pp. 258–272. Norwood: Abelex Pub. Co.

Hoyles, C. Noss, R. and Sutherland, R. (1991) *Final Report of the Microworlds Project.* University of London: Institute of Education.

Hughes, J. et al. (1995) The role of ethnography in interactive systems design. *Interactions* **2**, pp. 56–65.

Johnson, D. C. Cox, M. and Watson D. M. (1994) Evaluating the impact of IT on pupils' achievements. *Journal of Computer Assisted Learning*, **10**, pp. 138–156.

Jones, V. (1996) Classroom management. In Sikula, J. (ed.) *Handbook of Research on Teacher Education*, pp. 503–521. New York: Macmillan.

Kemmis, S. Atkin, R. and Wright, E. (1997) *How Do Students Learn?* University of East Anglia: Centre for Applied Research in Education.

Koedinger, K. R. and Anderson, J. R. (1993) Reifying implicit planning in geometry: guidelines for model-based intelligent tutoring systems design. In Lajoie S. P. and Derry S. J. (eds.) *Computers as Cognitive Tools*, pp. 15–45. Hillsdale: Lawrence Eribaum.

Laborde, J.-M. (ed.) (1995) *Intelligent Environments: the Case of Geometry.* NATO ASI Series F Subseries Advanced Educational Technology. London: Springer.

Laurillard, D. (1990) Computers and the emancipation of students: giving control to the learner. In Boyd-Barret O. and Scanlon E. (eds.) *Computers and Learning*, pp. 64–80. Wolinkham: Addison Wesley & The Open University.

Laurillard, D. (1993) *Rethinking University Teaching: a Framework for the Effective Use of Educational Technology.* London: Routledge.

Open Learning Technology Corporation Limited (1995) *Learning with Software*. On-line at www: http://gwis2.circ.gwu.edu:80/~kearsley/.

Lowther, D. and Sullivan, H. J. (1994) Teacher and technologist beliefs about educational technology. *Educational Technology Research and Development*, **42**, pp. 73–87.

Marton, F. (1994) On the structure of teacher's awareness. In Carlagen, I. Handal, G. and Vaage, S. (eds.) *Teachers' Minds and Actions: Research on Teachers' Thinking and Practice*, pp. 28–42. London: The Falmer Press.

McConnell, D. (1994) *Implementing Computer Supported Cooperative Learning*. London: Kogan Page.

Mellar, H. et al. (eds.) (1994) *Learning with Artificial Worlds: Computer Based Modelling in the Curriculum*. London: The Falmer Press.

Mercer, N. and Scrimshaw, P. (1993) Researching the electronic classroom. In Scrimshaw P. (ed.) *Language, Classrooms and Computers*, pp. 184–191. London: Routledge.

Mercer, N. (1993) Computer-based activities in classroom contexts. In Scrimshaw P. (ed.) *Language, Classrooms and Computers*, pp. 27–39. London, Routledge.

Olson, J. (1988) *Schoolworlds/Microworlds: Computers and the Culture of the Classroom*. Oxford: Pergamon Press.

Potts, C. (1993) Software engineering research revisited. *IEEE Software*, September, pp. 19–28.

Reiser, R. A. and Dick, W. (1990) Evaluating instructional software. *Educational Technology Research and Development*, **38**, pp. 43–50.

Reusser, K. (1993) Tutoring systems and pedagogical theory: representation tools for understanding, planning and reflection in problem solving. In Lajoie, S. P. and Derry, S. J. (eds.) *Computers as Cognitive Tools*, pp. 143–177. Hillsdale: Lawrence Erlbaum.

Sandholtz, H. J. Ringstaff, C. and Dwyer, D. C. (1997) *Teaching with Technology: Creating Student Centered Classrooms*. New York: Teachers College Press.

Schwartz, J. L. Yerushalmy, M. and Wilson, B. (eds.) (1993) *The Geometric Supposer: What is the Case of?* London: Erlbaum.

Soloway, E. and Pryor, A. (1996) Using computational media to facilitate learning. *Communications of the ACM*, **39**, pp. 83–109.

Sommerville, I. (1989) *Software Engineering*. 3rd ed. Addison Wesley.

Squires, D. and McDougall A. (1994) *Choosing and Using Educational Software: a Teachers' Guide*. London: The Falmer Press.

Squires, D. (1996) Production of educational software. In Plomp, T. and Ely, D. E. (eds.) *International Encyclopedia of Educational Technology*, pp. 217–221. Oxford: Elsevier Science–Pelgrum.

Taylor, R. P. (ed.) (1980) *The Computer in the School: Tutor, Tool, Tutee*. New York: Teachers College Press.

Tolhurst, D. (1992) A checklist for evaluating content-based hypertext computer software. *Educational Technology*, March, pp. 17–21.

Watson, L. (1993) Appropriate tools? IT in the primary classroom. In Beynon, J. and Mackay, H. (eds.) *Computers into Classroom: More Questions than Answers*, pp. 78–91. London: The Falmer Press.

Winograd, T. and Flores, F. (1986) *Understanding Computers and Cognition: a New Foundation for Design*, Reading, Massachusetts: Addison-Wesley.

Winograd, T. (1995) From programming environments to environments for design. *Communications of the ACM*, **38**, pp. 65–74.

Winship, J. A. (1989) *Information Technology in Education: the Quest for Quality Software*. Paris: Organisation for Economic Co-operation and Development.

Zahner, J.E. et al. (1992) Evaluating instructional software: a simplified model. *Educational Technology Research and Development*, **40**, pp. 55–62.