

## *Educating reflective systems developers*

Lars Mathiassen\* & Sandeep Purao<sup>†1</sup>

\*Department of Computer Science, Aalborg University, Frederik Bajers Vej 7E, 9220 Aalborg Ø, Denmark, email: larsm@cs.auc.dk, and <sup>†</sup>College of Business Administration, Georgia State University, Atlanta, GA 30302-4015, USA, email: spurao@gsu.edu

**Abstract.** *Systems development research shows that practitioners seldom follow methods and that the competencies required for successful development of computer-based systems go well beyond those represented in contemporary methods. These insights make us question the role that methods should play in educating would-be developers. Pedagogical theories, such as situated learning and double-loop learning, complement these insights. Integrating the two, we argue that students need to complement the simplified accounts that methods express, with reflections on methods-in-use and on development practice in general. We present operationalizations of this idea in two quite different academic settings. Based on a retrospective analysis of our experiences in these settings, and a comparison and evaluation of the two approaches, we propose a number of lessons that can be used to improve the education of would-be developers.*

**Keywords:** Method, reflective practice, systems development education, systems development practice

### 1. MOTIVATION

The systems development discipline has, since its very start, been populated by a constantly growing number of methods. These methods are traditionally viewed as prescribing exemplar processes that practitioners can follow to obtain success or at least satisfactory outcomes. The well-known saying: 'If you have a hammer, the world looks like a nail' could be transcribed to remind us of this widely held assumption on the development of computer-based systems: 'If you have a method, you know how to develop systems successfully'. Through empirical research (Vitalari & Dickson, 1983; Tan, 1994), we have, however, during the same period built a considerable body of knowledge on practice, in general, and on the use of methods, in particular. A close inspection of these results paints a somewhat different and, to some extent, contradictory picture. These results contain important lessons for those who engage in the

<sup>1</sup>A large part of this manuscript was compiled during a visiting position appointment at the Institutt for Informasjonvitenskap, Agder University College, Kristiansand, Norway.

development of new methods. But, more importantly, they challenge all of us to rethink the concept of method and, in particular, to reconsider the role methods should and can play in practice.

We also experience a strong pressure to expand and reshape our educational efforts related to information technology (IT). One expression of this is the ongoing debate questioning approaches and institutional settings for the education of would-be IT professionals (Denning, 1992, 1997, 2000; Norman & Spohrer, 1996, Dahlbom & Mathiassen, 1997). Do our students acquire the competencies and skills that are required in today's practices? Do we use appropriate pedagogical approaches? Do the selected disciplines and institutional settings give the students a proper understanding of and attitude towards professional standards and codes of ethics? Even though these questions relate to the design of our educational practices and institutions in general, they do have particular implications for how we should teach systems development.

This paper merges these two concerns by addressing the following overarching research question. How do we integrate what we know about systems development practice and pedagogical theories into better education of would-be developers? Specifically, we ask the questions:

- What insights can we draw from research on the practice and pedagogy of information system development (ISD)?
- What are the specific implications of integrating these insights for educating would-be system developers?
- How can we organize educational efforts, respecting these insights, to educate better system developers?

Our argument draws on research on systems development practice, pedagogical approaches and experiences from two cases of teaching systems development. We start, in Section 2, by extracting insights from the existing body of knowledge on systems development practice. In Section 3, we look at educational practices and pedagogical approaches to extract corresponding insights. Based on this conceptual foundation, we outline in section 4 aspirations for educating reflective system developers together with specific recommendations for such educational practices. Section 5 then presents two approaches from quite different institutional settings to educate reflective systems developers. Section 6 evaluates, compares and contrasts these two approaches based on our recommendations. Section 7 concludes and encourages colleagues to adopt reflective pedagogical practices.

## 2. PRACTICE

Over the past three decades, we have built a considerable body of knowledge on systems development practice. Even though both practice and technology have evolved and research approaches have changed, a number of fundamental insights into the nature of developing computer-based systems have emerged.

## The role of methods

Necco *et al.* (1987) reported, based on results from a survey, that the key factors in developing improved computer-based systems are improved involvement and better personnel. Methods are thus not considered a primary source for improving present practices. When they are considered, they are not aimed at improving the performance of specific activities but rather improved organization of the overall process including shortening the lifecycle. Turning from beliefs in methods to their actual use, Bansler & Bødker (1993) reported (from a field study of three organizations making extensive use of structured analysis) that the systems developers only use the method partially, and that the parts they use differ across organizations and projects. The developers carry out their work in ways that deviate from the prescribed procedures in several major ways. They conclude that experienced developers 'pick and choose among the various formalisms given in the method, adapt them for their own purposes, and integrate them into their own design processes' (p. 189). There are, thus, important differences between methods, i.e. prescriptions found in the professional literature, and their use, i.e. the processes through which practitioners apply methods to practical problems. Going one step further and focusing on the actual benefits of using methods, interesting results are reported by Guindon *et al.* (1987), who analysed think-aloud protocols of three professional developers. They conclude that lack of knowledge of the application domain and specialized design schemas is the primary obstacle to successful design. Lack of methodological knowledge on how to structure the design process is also an obstacle to successful design because it can lead to poor allocation of resources during the various design activities. But it is of secondary importance. In other words, it is more important to have specialized knowledge about problems and possible solutions than it is to have general knowledge on how to structure and conduct development processes.

## Competencies required

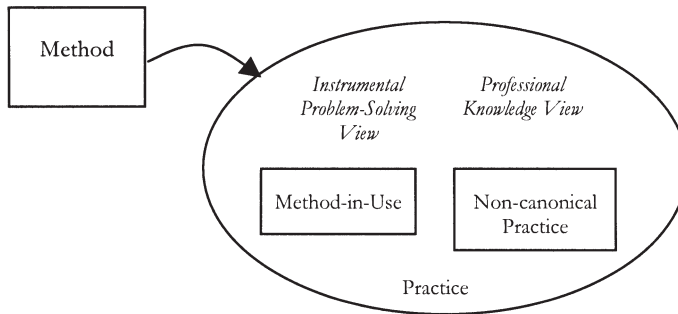
To understand better the important, but limited role that methods play as guidelines for practice, we look more closely at studies that have explored the competencies required for systems development. The first empirical study of required competencies was reported by Shrouf (1970) (quoted by Vitalari, 1985). Her survey indicated that organizational and administrative competencies are perceived as more important than technical and computer-related competencies. Vitalari (1985), based on the analysis of protocols of 18 systems analysts, found that they primarily exercise knowledge related to aspects of the computer system, relegating behavioural and organizational issues to a less dominant role. The two studies thus suggest important differences between which competencies are perceived as important by managers and analysts (Shrouf's study) and which competencies are exercised by analysts (Vitalari's study). White & Leifer (1986) studied competencies that project team members perceived as leading to systems development success. Their findings suggested that the required competencies may vary during the project, and that the top five competencies perceived as impacting systems success across the development phases are: business knowledge, good

communication skills, technical expertise, analytical skills and good organizational skills. Along with the findings of Shrouf (1970) and Vitalari (1985), this study indicated that a broad range of competencies is required to develop computer-based systems successfully. The required competencies range from social to technical and address both process and system issues. Other complementary studies suggest the kinds of behaviour that lead to success. Vitalari & Dickson (1983) compared the behaviour of high- and low-rated systems analysts. Based on an identification of qualitative differences in the problem-solving methods of the two groups, they suggested that there are at least four different behaviours that characterize successful design. These include analogical reasoning, the setting of goals and formulation of strategies while maintaining flexibility, managing of emerging hypotheses and actively dealing with the interface between analyst and user. Additional insights were provided by Tan (1994) about the communication behaviour of systems analysts working with users to determine requirements. Her findings suggested that effective interaction is related to the strengths of the interpersonal relationship between analysts and users. Effective communication is thus the outcome of complex processes that are influenced by the personal and situational characteristics of the participants.

### Insights about ISD practice

The above studies suggest that methods do play several important roles as guidelines for practice. They help in structuring the development process, they offer a set of modelling techniques and tools, and they constitute a common language for communicating about systems development across activities and projects. But we see in these findings that the role played by methods is in many ways limited. Lack of methodological knowledge is not the primary obstacle for success, methods are typically used in a selective and highly pragmatic fashion, and methods are, as a consequence, not considered the primary means for improving practice.

The studies also show that successful behaviours go well beyond what is described in traditional methods. Success is intrinsically related to the analyst's ability to understand, deal with and actively form relationships with users, dictated by the specifics of the situation in question (Fayad, 1997). These findings within systems analysis find broader support in the extensive empirical studies of systems development (Elam *et al.*, 1987; Guindon *et al.*, 1987; Krasner *et al.*, 1987; Curtis *et al.*, 1988; Waltz *et al.*, 1993). These researchers found that experienced systems developers primarily make use of a repertoire of past experiences and adapt the best fitting of these to the present design situation (Guindon *et al.*, 1987). They also found that context-sensitive learning is important and that much of the information that needs to become part of a team's memory is not captured formally in the project documentation. Curtis *et al.* (1988), for example, saw the development of computer-based systems as a learning, communication and negotiation process, calling for environments to become a medium for communication to help integrate people, tools and information. Waltz *et al.* (1993) recommended active promotion of acquisition, sharing and integration of knowledge between team members.



**Figure 1.** The practice of systems development.

These results negate a simplistic understanding of the relation between methods and practice in which systems development is guided by technical rationality. The mechanistic, rational view implies a form of professional practice in which developers start with given objectives and choose optimal methods to realize them. In doing so, they apply scientific knowledge to guide their choices and to perform the required tasks. Viewed in this way, practice is simply the result of using methods that are based on scientific knowledge about the discipline. Research on systems development reveals a different picture (Figure 1).

Methods are guidelines for practice and expressions of espoused theories on systems development. They are quite different from methods-in-use, understood as those parts of ISD practice in which practitioners apply methods to practical problems. Moreover, methods-in-use constitute only one part of practice. The non-canonical practices in which developers deal with the uncertainty, instability, uniqueness and value-conflict of the situation at hand are at least as important for the success of a systems development project. These parts of the ISD practice are not and, in most cases, cannot be explicated or codified as is the case with the method-in-use parts. We should therefore distinguish methods from methods-in-use, thereby explicating an instrumental problem-solving view that focuses on how methods are used. We should also distinguish methods-in-use from non-canonical practice, thereby adopting a broader, professional knowledge view that includes the tacit and difficult to explicate elements of ISD practice.

### 3. EDUCATION

As systems development technologies and practices have changed over the years, so have our efforts and emphases for educating systems developers. Although different fields such as computer science, software engineering and management information systems have undertaken this task, the models of education practised across these disciplines show a number of shared themes.

## Recurrent themes

Of late, considerable efforts have been devoted to standardizing course content (Couger *et al.*, 1995; Leidner & Jarvenpaa, 1995, Bourque *et al.*, 1999; Parnas, 1999). The 'system design' component of the software engineering curriculum recommended by SWEBOOK includes knowledge elements such as general design concepts, design processes, architectural structures, quality analysis and evaluation, design notations, design strategies and methods, design tools and standards (Bourque *et al.*, 1999). (The curriculum includes components such as configuration management, construction, design, engineering infrastructure, engineering management, engineering process, evolution and maintenance, quality analysis, requirements analysis and testing.) The curriculum recommended by ACM/DPMA (Couger *et al.*, 1995) includes systems development lifecycle activities, communication with users and a variety of specific tools, techniques and approaches. The emphasis is clearly on ensuring that the information systems developer will, on completion of the education, 'possess' certain 'knowledge' about a number of 'topics' including a number of prescribed 'methods'. Recent surveys of systems development courses (Russo & Mistic, 1999) show that, to a large extent, the actual coverage matches the recommendations above. Their work, which builds on previous work by Trauth *et al.* (1993), is based on a survey of teachers listed in the MIS Faculty Directory (MISRC, 2000). They found that, in order of priority, teachers emphasize: (1) defining new system requirements; (2) defining the scope and objectives of systems; (3) preparing specification and requirements reports; (4) analysing existing systems; and (5) developing system models. The bias towards a prescribed, rational, top-down approach to systems development and an emphasis on documentation is evident in these.

Turning from course content to approaches to instruction, no standardization efforts (perhaps rightly so) have been undertaken for approaches to instruction in systems development. A few surveys, personal accounts and experience reports are, however, available, which provide glimpses of approaches used to educate would-be systems developers. Participation in a team-based systems development project is, perhaps, the most frequently used approach. Hagan *et al.* (1999), for example, described the systems development capstone project that students experience as they participate in the building of a real system for a client outside the university. They describe an emphasis on management issues and development process with deliverables that reflect this emphasis. Clark & Boyle (1999) suggested that ISD projects should be seen as an opportunity for students to engage in realistic activity, which allows them to learn something about the nature of the discipline. Schenk *et al.* (1998) articulated well the belief in project-based approaches as 'every instructor knows that . . . teaching students the characteristics of a method is insufficient to enable them to apply the methodology.' Russo & Mistic (1999) described different 'activities' undertaken by teachers in systems development courses. These include, in order of emphasis, working in teams, drawing system diagrams, practising system development techniques (e.g. JAD), making presentations and submitting reports. The team-based systems development project is used as the core vehicle to realize the above activities. The activities emphasized least by the educators and, interestingly, valued

most by practitioners include critical analysis of case studies and alternative designs, and exams that require the analysis of a situation (Russo & Mistic, 1999).

The brief review above suggests that, as educators of systems developers, we hold certain assumptions that guide our stance to teaching and how we operationalize it into specific activities. These assumptions appear to match well the rationalist version of assumptions articulated by Russo & Stolterman (1998). As educators, we believe that 'methods are useful' and that 'designers' behaviours can be changed for the better by educating them in the use of these methods'. Although working in teams is considered important, the educators appear to consider it as simply a natural ingredient of the project experience, without giving much further thought to exploiting the team-based work environment.

### Learning theories

The summary above requires an investigation into learning theories that we, as educators, may be implicitly drawing from to create these approaches. Such an inquiry would reveal to us how we perceive the construction and communication of knowledge about systems development. This reverse-engineering of learning theories can be difficult without explicit statements from educators involved in the process or in-depth studies of teaching accounts. However, based on the analysis of personal experiences, written accounts and surveys such as the ones outlined in the previous section, we can attempt this discovery. Two groups of theories can be considered as candidate learning approaches underlying current education practices, the constructivist and the social/cultural [apart from the early stimulus-response camp (Skinner, 1968), which may be considered as the basis of the lecture method of teaching (Leidner & Jarvenpaa, 1995)].

The first group of theories (the constructivist camp; Piaget, 1929; Bruner, 1966) assumes that each individual constructs his or her own reality of the objective world (Yarusso, 1992; Leidner & Jarvenpaa, 1995). The theories in this camp therefore argue that learning is an active process in which students construct new ideas or concepts, while engaged in new experiences, based on their current and past knowledge (Bruner, 1966). The goal of learning is to allow the formation of abstract concepts (O'Laughlin, 1992), not feeding these externally to the student. The concept of cognitive structures is therefore central (e.g. Piaget, 1929) to this group. Another set of theories in this camp emphasizes the 'group' over the individual. Conversation theory (Pask, 1975) suggests 'teachback' as the critical approach, where learning occurs through conversations about a subject matter. Vygotsky (1978) suggested that interaction plays a key role in the development of cognitive structure.

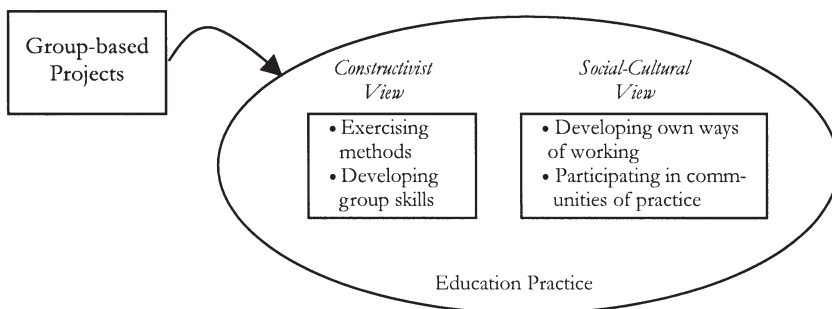
The second group (the social/cultural camp; e.g. Lave, 1988) disagrees with the constructivist view that the goal of learning is the formation of abstract concepts. They emphasize that knowledge cannot be divorced from the historical and cultural background of students (O'Laughlin, 1992; Leidner & Jarvenpaa, 1995). An important theory in this camp is situated learning (Lave, 1988), which argues that learning as it normally occurs is a function of the activity, context and culture in which it occurs (i.e. it is situated). Students become involved in

a 'community-of-practice', where they progress from beginner to expert. Lave & Wenger (1990) called this process, which they argue is unintentional rather than deliberate, 'legitimate, peripheral participation.' Brown *et al.* (1989) emphasized the idea of cognitive apprenticeship, emphasizing active perception over concepts and representation. Argyris (1976) suggested double-loop learning, in which participants can learn to change underlying values and assumptions while attempting to solve complex and ill-structured problems. The knowledge constructed is thus highly local, specific to a context, i.e. situated, and appropriate for the individual.

### Insights about ISD education

Current practices in educating systems developers focus on the use of group-based projects to support experiential learning and to promote social interaction among team members, which is expected to make knowledge about system development more concrete. Through lectures and other classroom activities, the educator enhances and solidifies this knowledge. Projects are thus used to force the discovery of abstract concepts that underlie methods. The use of team projects appears to be dictated by the conviction that the human issues are more important than the technological ones, and by the desire to improve communication and listening skills (Leidner & Jarvenpaa, 1995). The pedagogical positions that underlie these practices may therefore be identified as primarily constructivist. Although project-based work allows the opportunity, important aspects of learning that occur in a situated environment (Suchman, 1988), from a social/cultural view, are rarely exploited. For instance, experiences that occur while working on a project are neither articulated nor shared, and individuals are rarely encouraged to reflect on their experiences to discover situated knowledge. Context-specific ways of working and the creation of situated knowledge, although often encountered and used by the project participants, are rarely articulated or supported. Elements of learning such as peripheral participation in a community of practice are therefore not actively supported.

Figure 2 outlines the practice of educating systems developers, specifically how group-based projects are used to serve complementary goals. The projects can be used in a constructivist way to exemplify prescribed ways of working, promote the discovery of abstractions



**Figure 2.** The practice of educating developers.

and provide opportunities for improving communication and listening skills. They can and should, however, also be used in a social/cultural way to develop the participants' ways of working as professionals through their active and reflective participation in the communities of practice that emerge.

#### 4. IMPLICATIONS

In this section, we argue, following insights from the previous two sections, that our aspiration as teachers should be to educate reflective systems developers. Our argument includes specific recommendations for leveraging appropriate educational approaches that will ensure that students not only learn methods, but also discover their own methods-in-use and reflect more broadly upon their experiences as members in a community-of-practice (Brown & Duguid, 1991).

##### Aspiration as educators

Professional practice as we have come to know it through empirical research can be seen as a particular instance of reflection-in-action (Schön, 1983; Mathiassen, 1999). The underlying rationality of developer behaviour, the roles methods play in practice and the competencies required to deal with the challenges in systems development are explicated through this perspective. Systems developers must therefore bring to bear something more than a repertoire of general methods and tools. They must engage in reflections and dialogues to generate the necessary insights into the situation at hand (Mathiassen, 1999, p. 68). This view of systems development as a reflective practice is supported by a number of sources. Lanzara (1983) argued that systems development is not merely an instrumental problem-solving activity. He emphasized the important role played by previous experience and by the problem setting in each new development situation. Stolterman's studies of how designers think about practice and methods revealed that rationality in practice emerges as a result of the confrontation between the designer and the specific design situation and is quite different from the abstract and simplistic rationality of methods (Stolterman, 1991, 1992). Bjerknes (1989, 1991) suggested more specifically that the tensions and contradictions involved in specific project situations constitute an important source of insight that can help to form and manage systems development projects.

A theory of learning that lends support to this view of systems development practice is the theory of situated learning (Lave, 1988; Brown *et al.*, 1989), which asserts that learning occurs as gradual acquisition of knowledge and skills when an individual is immersed in an authentic environment. Reflection and reflection-in-action signify learning by doing in the sense of both dealing with the uniqueness involved in each new development situation and continued learning and knowledge building. Such situated learning allows the student to acquire skills that are not amenable to articulation as abstract concepts. The skills therefore have a highly situated character and can only be acquired over time, in an unintentional rather than a deliberate manner. Social interaction is an important element of this process that facilitates the

acquisition of behaviours and beliefs intrinsic to the ways of working. Accordingly, Brown *et al.* (1989) advocated an epistemology for learning that emphasized active perception over concepts and representation.

We argue therefore that our aspiration should be to educate our students in ways that help them become reflective systems developers. Although this may be seen as a consequence of the arguments presented so far, it raises a series of questions that are considerably more difficult to answer: What is then the role of methods in educating systems developers? How can situated learning become part of the learning process? How can we use projects based on realistic problems and still maintain the focus necessary to provide the students with the required skills?

### **Recommendations for educating developers**

We propose a number of recommendations for educating reflective systems developers, drawing upon the relationships between methods, methods-in-use and non-canonical practices and instantiating the constructivist and social/cultural themes for pedagogical approaches.

**Methods should be used as training programmes.** We recommend that methods be viewed as training programmes (Mathiassen *et al.*, 1996) instead of simplified or idealized form of practice. This view allows teachers to consider methods in a different light, opening different ways to use the methods during education. The view presented by Naur (1985) appears closest in spirit to the above. He used a theatre metaphor to explain the role of methods in practice, indicating that it is neither that of the author nor that of the conductor. Instead, methods play the role of the prompter by helping us to remember that which has momentarily slipped our mind. The prompter (and the manuscript) play more prominent roles during rehearsals than they (hopefully) do during the performance of the play. Considering methods as virtual training manuals offers a more constructive perspective on their use in educating system developers. Methods are not idealized practices to be imitated by students. Methods provide an overview and explanations of a set of related concepts, notations and techniques. Each of these are taught in dedicated sessions in which students can practise to build relevant skills. The students cannot practise a method as a whole in such dedicated sessions. Instead, they learn the method by adapting it to projects. Opportunities to practise elements of method in this manner reflect adaptation of the constructivist approach to pedagogy (O'Laughlin, 1992), where the student is encouraged to engage in the active process of applying the method to allow the construction of abstract concepts.

**Projects should be designed as laboratories for discovering methods-in-use.** Methods-in-use are considerably more important to successful practitioners than the simplified accounts of methods, that is 'it is more about skills than it is about methods' (Bach, 1999). The elements of methods should be practised and the underlying rationale understood, but the education should allow progression from learning methods to the discovery of methods-in-

use. One possible approach to achieving this is the use of projects through the term. Such projects should encompass multiple iterations and reviews of these iterations to ensure that there are opportunities to discuss how different components of methods are applied and how this application changes. Projects should address relatively familiar domains and contain demonstrably appropriate realism and complexity to promote a feeling of authentic experience to the students. The levels of realism and complexity should approach what we may expect in professional practice. They should necessarily fall short of it though to provide opportunities for reflection and to ensure that the teachers can participate in more meaningful ways to coach groups during their progression through the project. The authentic but limited perspective should promote sufficient time and opportunities so that intra-group interactions can flourish to promote better learning. In this manner, the interactions should be allowed to play a significant role in the formation of cognitive structures as suggested by group-oriented sets of theories (Vygotsky, 1978) in the constructivist camp of pedagogical theories.

**Projects should provide appropriate settings for building collaborative skills.** In addition to realism and complexity, projects should contain sufficient ambiguity to allow individuals and groups to progress in different directions and hence require the members to co-ordinate their efforts. Ambiguity promotes greater levels of interaction between group members because of the need to test assumptions and solution approaches, which could vary greatly depending upon the different perspectives that each group member may bring. The different perspectives will also make it imperative on the group members to arrive at a negotiated understanding of the problem that their project addresses and of the work practices that they share. Systems development is a highly collaborative activity, and it is important that the education offers settings for building collaborative skills. The highly situated nature of knowledge (Lave, 1988) that emerges from this collaborative activity suggests a clear mapping to the social/cultural pedagogical approaches.

**Students should reflect critically on the communities of practice within the projects.** There are several elements of non-canonical practice that cannot easily be expressed but are somehow 'held in the head' (Bach, 1999) and appear to work in highly local contexts. These represent an important form of expertise for systems developers, which must be learned from communities-of-practice (Brown & Duguid, 1991). Communities-of-practice cannot, however, be designed. They emerge as expressions of the problem-solving and collaboration culture within and between the project groups. Undergoing the project experience together builds a community of practice within a group. Subscribing to the values and aspirations of the material provided by the educator contributes to the building of a community of practice between groups, as do work practices and team member behaviours, desirable or not. Developing an ability to recognize and acquire these should be part of how the project experiences are leveraged. The role of coach and supervisor is therefore essential in making the students reflect critically on their experiences. The students should learn from the non-canonical practices that are essential in systems development, and they should develop their own critical sense of

what constitutes good and useful as opposed to inappropriate practices. The idea of cognitive apprenticeship (Brown *et al.*, 1991) from the social/cultural pedagogical camp maps well with the roles that student and educator should play following this recommendation. The activities can also be described in terms of double-loop learning as explicated by Argyris (1976).

**The role of methods should evolve.** Professionals do not discard methods in one swoop. Rather, their use of methods undergoes a gradual transformation as they learn the methods, begin slowly to adapt them and, eventually, formulate their own views of methods, that is, for the experts, 'it is more about becoming better than it is about being good' (Bach, 1999). A similar evolution of the role of methods should be facilitated in education as the students study methods, practise sets of concepts, notations and techniques, and engage in projects. The use of multiple iterations and reviews of project deliverables from other teams should provide students with specific opportunities, whereby such changes can be observed and discussed either among the group members or with the educator. The gradual evolution can be viewed as the result of legitimate, peripheral participation in the community of practice suggested by Lave & Wenger (1990), leading to an unintentional rather than deliberate learning process suggesting instantiation of the social/cultural pedagogical approach.

Although these recommendations may be appealing, their usefulness will be limited unless they can be operationalized into specific teaching activities. In the next section, we discuss two different approaches, which, in their own way, have attempted to operationalize these recommendations.

## 5. APPROACHES

We illustrate our discussion by presenting two education approaches from quite different institutional settings. The first approach is evolutionary in the sense of being implementable in most teaching environments. The other approach is more radical.

### An evolutionary approach

The first approach comes from a North American university course entitled Object-oriented specifications. The primary audience is Masters level (graduate) students, typically enrolled in a Master of Science degree programme in Computer Information Systems. The intention is to introduce students to the analysis of business requirements as objects and to the creation of object-based information systems applications. The course is a traditional term-based one. The class meets once or twice a week for a specified period of time. The in-class sessions are filled with discussions of concepts and their application to solving small problems. To emphasize the importance of iteration and to demonstrate the messy nature of the process, the teacher conducts two sessions in class where he plays the role of the designer, talking through the act of design. One session is spent creating use cases, and the second is devoted to constructing a class diagram.

**Projects.** During the course of a term, the students work on two team projects. The teams are first given a simple, reasonably well-defined set of requirements from a domain that is not difficult to understand, such as an IT consulting company or an IT training company. The project is not very large to ensure that the students can work through the entirety of the problem without feeling overwhelmed. The students use a CASE tool to create appropriate use cases and a class diagram. The project lasts for about 25% of the duration of the term.

As the students work on the project, they are encouraged and required to send reflections to their team members and the teacher in an informal, although timely manner. Each Monday morning, every team member sends an email to all team members, indicating the progress they have made so far in their project, the problems they faced and how they resolved the problems. The students are encouraged to discuss problems dealing with understanding the material presented in class, problems dealing with applying the material to the project and concerns about teamwork. The reflections constitute the team diary. A first draft version of the specifications is reviewed by the teacher and another team. Each team also gets a chance to study the specifications and reviews of a third team. Based on the feedback from the teacher, the review from another team and their exposure to the commented-upon-draft from a third team, each team now finishes the project and prepares the final version. The weekly reflections continue.

The second project is considerably larger and ill-defined. The students are given a brief (5–7 line) problem description and a few interview transcripts that shed some light on the requirements. In the past, the case has dealt with an airline company, with each group tackling different parts, such as flight scheduling, repairs and maintenance, catering and customer service. The students conduct interviews with individuals from local companies or with the teacher, who plays the role of an employee of the company. The weekly reflections continue. The project spans about 60% of the duration of the term. The draft is once again critiqued by another team and given feedback on by the teacher, and the draft from at least one more team is made available to each team. A CASE tool is used to document the complete project and to co-ordinate activities between team members. The final deliverable includes use cases, class diagrams, state transition diagrams, interaction diagrams and architectural decisions.

The results are judged by a panel of experts called jurors. The jury represents a practice commonly used in architecture. The teams display their projects in class, taped to the classroom walls. Different team members are then assigned as jurors to other teams, along with external jurors drawn from departmental faculty, doctoral students, visiting faculty, past students and local IT professionals. The teacher facilitates a jury process that opens for the informal exchange of ideas among teams and team members and allows time to reflect on the different approaches taken to address the project by the different teams.

**Learning.** This evolutionary approach provides at least four different opportunities for learning about and reflecting upon the craft of systems development. First, as the teacher demonstrates the messy nature of design in class, the students watch and learn how a designer

reflects-in-action. They reflect on the teacher's reflection-in-action and understand that systems development practice requires a set of skills different from the simple understanding of concepts. The second opportunity is presented when the students, as team members, reflect on their project every week and articulate the issues they face and how they resolve these. Here, they learn to deal with a team of systems developers who may have different perspectives on the problem. They learn to see that there can be different, equally valid viewpoints on design. Issues such as team conflicts, resource management and work distribution surface during this time. A third opportunity is presented when the students review other teams' projects. Typically, as the design process results in specifications that build on a number of assumptions, the review can bring these assumptions to the surface. This is a difficult result to achieve without the benefit of an external perspective. The review also provides the students with a necessary incentive to learn from their own as well as their competitors' mistakes. Finally, the fourth opportunity is presented through the design jury. Here, the students get to see the end-product of all the other teams. Given a similar set of requirements, it is interesting to see the widely different design perspectives that are brought to the project by the different teams.

### **A radical approach**

The second approach comes from a European university course entitled System analysis and design. The course is an integral part of the third semester of a Masters degree in computer science. During the first two semesters, the students follow a basic programme together with other science and engineering students. The semester in question is then the first (out of six) in which the students specialize for a Masters degree in Computer Science. The overall theme is 'Development of a computer system', and all students follow the same activities. They spend half their time following four courses entitled Programming, Systems analysis and design, Algorithms and data structures, and Network and computer architecture. They spend the other half on one large project in groups of five to seven. Each group has a supervisor and shares an office at the university. The courses in Programming and Systems analysis and design support the projects directly. They provide the students with the basic skills and competencies needed in the project, they are taught in parallel to the project, and the students are graded in these courses based on their project. The students should build and demonstrate competence in analysis, design and programming of a computer system for a specific use context. The projects are evaluated based on practical criteria (the practical part of the project) and academic criteria (the academic part of the project). A group could, for instance, develop a computer system for a video rental shop (the practical part) and, based on their experiences, reflect upon the role of specifications in systems development (the academic part).

**Projects.** Each course is organized into 15 4-h sessions. Each session consists of a lecture followed by group discussions and exercises, typically related to the project. The courses in Programming and Systems analysis are condensed in the first half of the semester with two

sessions per week and sometimes with intensive 2-day workshops that focus on a coherent set of techniques. The groups are formed, and the projects are outlined during the first week of the semester. Each group is encouraged to design its own project within the overall theme of the semester. They contact the persons or organizations that serve as clients and future users, and select and tailor methods, techniques and tools to suit their needs. The analysis, design and programming activities are performed as an iterative process supported by formal reviews. Faculty members conduct the first review, while another group conducts the second. Both reviews are scheduled from the start of the semester. Deadlines cannot be negotiated, but the groups can modify requirements, specifications and deliverables in collaboration with their supervisors. Finally, they select the perspectives that they will apply to reflect critically on their experiences in the study report.

The two courses and the project are evaluated based on the system and the system documentation (the practical part) and the study report (the academic part). The evaluation is organized as a 3-h session between the group, the supervisor and an external examiner. First, the group makes an oral presentation of their project (0.5 h). Secondly, the system is demonstrated (0.5 h). Thirdly, a discussion is led by the supervisor based on the documentation and the study report (2 h). During this discussion, each individual student responds to issues raised. The students are graded individually based on a holistic view of their performance. As a guiding principle, the study report counts for 20%, the computer system with documentation counts for 50% and the presentation and discussion counts for 30%. The practical part of the project is evaluated based on the following practical criteria: (1) competence in analysis, design and programming must be documented; (2) a functioning computer system must be demonstrated; (3) the documentation should follow professional standards; and (4) the project should systematically apply methods, techniques and tools. The academic part of the project is evaluated based on different criteria: (1) the reflections must address experiences in organizing and conducting the project; (2) explicit argumentation for the choices involved is emphasized; and (3) the reflections must address experiences in applying methods, techniques and tools.

**Learning.** The radical approach provides at least four different opportunities for learning and reflecting upon the craft of systems development. First, by introducing part of the method in a dedicated 2-day workshop, the students get a first sense of how different techniques can be combined and merged to create coherent and useful results. During the workshop, a number of techniques are presented and then practised in subsequent sessions. They are all practised on the same problem, and the students are encouraged and coached to arrive at, say, a first version of a complete requirements document. The workshop ends with an exhibition in which the groups present their results on posters and exchange viewpoints and experiences across groups. The second opportunity is the development project experience in which students work on a realistic problem and interact with potential users. The students are required to design their own project, identifying and defining the problem, establishing contacts with users and making plans that fit with the given requirements. The third opportunity for reflection is the reviews performed in each project. In the first review, the students are con-

fronted with evaluations and recommendations from a teacher other than their supervisor. The supervisor acts as review leader as students learn how to conduct a formal review. In the second review, they themselves act as the reviewer of other groups. In that way, they are challenged to reflect upon their own project in a new light. Finally, the fourth opportunity for reflection is the study project, in which the students have to document an academic enquiry into their own experiences. They select a perspective of their own choice and present their experiences, evaluate them critically and contrast them with findings from the literature. The resulting study report is also evaluated and discussed in the final session with the supervisor and external examiner.

## 6. REFLECTION

In this section, we evaluate the two approaches – evolutionary and radical – against the recommendations outlined in section 4. We consider how different elements in each approach address the recommendations and offer a comparative evaluation of the cases highlighting similarities and differences between the two.

### **Methods should be used as training programmes**

The evolutionary approach contains several elements that promote practising of methods, using methods as training programmes. First, the teacher practises method elements in front of the students. It allows the students to reflect on how the teacher is practising and reinforces practising as an important element in learning the method. The selective focus of the teacher during these in-class sessions also allows the students to see how different components of the method may be practised. As the students work on the small project, which is familiar and scoped down, it allows them to focus on learning and practising elements of the method instead of dealing with the uncertainties of a real project environment.

The radical approach, on the other hand, uses a series of sessions in which specific concepts, notations and techniques are explained and demonstrated. Each session consists of a lecture followed by group discussions and exercises. The latter provide students with opportunities for practising method elements. When some of these lectures are organized into 2-day workshops, a virtual training camp is established without interruptions from other study activities. This increases the intensity of practising method elements and helps the students to understand the relations between the individual elements.

Both approaches use the methods as training programmes as they limit the students' experiences for the duration of practising. This is achieved by limiting the scope of the problems tackled and by having the teacher participate in the problem-solving process. Opportunities are therefore created that allow the students to reflect on their experience to develop a better understanding of the elements of the methods. Both approaches appear to follow this recommendation, although in quite different ways.

### **Projects should be designed as laboratories for discovering methods-in-use**

The in-class walkthroughs of method elements performed by the teacher with the evolutionary approach also perform another role. The students see the teacher's method-in-use preparing them to engage in the projects where they can discover their own methods-in-use. As they start with their first project, their world is somewhat restricted, allowing them to focus on bringing the method into practical use. The weekly reflection is then an opportunity to articulate and communicate their attempts at discovering the methods-in-use. This discovery is aided further by sharing reflections between team members, so they can see how others are discovering their methods-in-use.

In the radical approach, the study project that accompanies the development project requires that students reflect on issues related to the method and its practical use. As this is a stated requirement, the radical approach makes the discovery of the method-in-use more explicit than the evolutionary approach. The radical approach also allows a more intense discovery because the projects are seen as the primary driver for each semester and the teaching of method elements is not interspersed with the projects but organized in a separate course. The projects are truly seen as laboratories, which build on the practising of method elements.

Both approaches use projects that are relatively authentic and rich. The realism level is achieved by either having a project for which a real company acts as the client or involving people from real companies for requirements gathering. The domains from which these projects are taken are somewhat unfamiliar to the students. Finally, the project work in both approaches contains elements of iteration and intermediate deliverables, making the experience close to a real development project. The radical approach builds on this recommendation in a very explicit and elaborate fashion, whereas the evolutionary approach relies on using smaller projects within the course to help the students to discover and reflect upon their method-in-use.

### **Projects should provide appropriate settings for building collaborative skills**

In the evolutionary approach, the second project helps in leading students towards building more group and collaborative skills. The problem the students are required to tackle for this project contains several ambiguities. This requires the students to interact within the group. The relatively short case description, along with a few interview transcripts, provides them with sufficient degrees of freedom, which manifests as spirals of discussions in which they are forced to deal with different interpretations within the group. They often conduct interviews with people from real businesses as well, making the acquisition of listening and communication skills an important product of the project experience.

With the radical approach, the students tackle real-world problems, making interaction a vital component of the systems development project. In addition, as the group physically shares an office with relatively easy access to a supervisor, they develop a daily routine in

which co-ordination and communication play a crucial role. Also, as they are mutually dependent on each other for the whole semester, they have to address conflicts and problems related to the project, to the use of the method and to their social interaction in general. The students experience these varied interactions and need to develop good group skills to succeed with their projects.

Both approaches provide good opportunities for building collaborative skills, even though the evolutionary approach provides less intense experiences compared with the radical approach.

### **Students should reflect critically on the communities-of-practice within the projects**

The evolutionary approach requires students to reflect on their experiences on a weekly basis. Here, the students are forced to articulate the experiences they are undergoing during the project. More importantly, these reflections are shared among the group members throughout the term, allowing students to examine how others in the team perceive the same experiences in different ways. The reflections contain not only issues that concern the use of methods but also several others that address non-canonical practices.

The more intense project experience provided by the radical approach allows a higher level of participation, and the study project (that accompanies the development project) explicitly requires the students to reflect critically and contrast with state-of-the art knowledge on systems development. Both approaches use iterations and emphasize intermediate deliverables to ensure significant interaction with the teacher or supervisor playing the role of coach. In this way, each project group forms its own community-of-practice. The overall set-up and the reviews allow the formation of a larger community-of-practice, in which the students and groups can air and test their hypotheses.

The radical approach may provide a qualitatively better experience and opportunity to reflect on the community-of-practice as a result of the explicit requirement of a study project, but the evolutionary approach also provides sufficient opportunities to engage critically in reflections on the community-of-practice involved.

### **The role of methods should evolve**

Considering this recommendation across the two approaches, we can identify a number of different ways in which methods are used as learning progresses:

- 1 The elements of the method are presented.
- 2 The elements are demonstrated with the use of small examples.
- 3 A limited experience is engineered to allow students to practice method elements.
- 4 The method is used in a larger, less specified project that involves uncertainty.
- 5 Students are exposed to intermediate results from other groups.
- 6 Students are required to reflect on non-canonical aspects of their own practice.

7 Evaluation sessions allow students to see the limited role that methods play in developing good solutions.

The role of methods evolves in both cases, but there are variations and differences in the individual steps taken. Both approaches do, however, follow a pattern similar to the one described above as the education progresses.

## 7. CONCLUSION

The source of ideas discussed in this paper is the rich stream of research on systems development, emphasizing the importance of methods-in-use and non-canonical practice. The learning theories we have discussed suggest plausible approaches that map against these requirements for learning about systems development. Our recommendations, in fact, integrate lessons learned from these two perspectives. An emphasis on professional knowledge as knowing-in-action, with its attendant ontological requirements, is at the core of our recommendations. Reflection-in-action is therefore the epistemological stance that we have advocated.

The notion of a practicum explicated by Schön (1987) comes closest in spirit to our recommendations. A practicum is 'a setting designed for the task of learning a practice, . . . a virtual world, relatively free of the pressures, distractions and risks of the real one, to which, nevertheless, it refers, . . . a collective world in its own right, with its own mix of materials, tools, languages and appreciations' (Schön, 1987, p. 37). A professional discipline such as systems development can benefit from these powerful ideas. Turning the ideas into action and making the perspectives appropriate for a discipline is, however, a challenge. In this paper, we have made an effort to operationalize the notion of a practicum in the context of teaching systems development.

The two approaches we have described represent possible paths that we may adopt as educators to progress towards this goal. The evolutionary approach represents a purely local initiative within the framework of one particular course. Hence, it would be practical for most of our colleagues. In contrast, the radical approach represents an initiative that requires one coherent semester, in which courses and one large project are designed as a single unit of education. The latter may, as we have argued, provide qualitatively better options for reflective teaching because of the explicit requirement of a study project, but is more difficult to implement. It is conceivable that there are other approaches to achieving these goals. The ones we have described represent our attempts at integrating these ideas into our teaching.

There are a few caveats that we offer to colleagues, who may wish to follow our recommendations. The richness of reflection-in-action and the learning that can occur as a result depends upon the level of interest and commitment demonstrated by the students. In particular, our recommendations require the students to be motivated and willing to engage in meaningful interactions within as well as across project groups. There is a real danger that

project groups comprise students, who, instead of engaging into spiralling discussions, may degenerate into using a naïve task decomposition strategy. Ensuring excellence from the teachers as well as demanding it from the students is therefore a prerequisite to making these ideas work. In striving to achieve this, it becomes necessary to turn the lens onto our own practices as teachers. As we engage in educational practices, we should use reflection-in-action as a tool continuously to improve our teaching. We hope the present discussion has served as an inspiration to start doing so.

## ACKNOWLEDGEMENTS

We would like to thank Mark Keil, Hans Olav Omland and Arun Rai for their comments on an earlier version of the manuscript. We acknowledge the thorough comments from reviewers and the associate editor, which have helped to refine and focus the paper further.

## REFERENCES

- Argyris, C. (1976) *Increasing Leadership Effectiveness*. John Wiley, New York.
- Bach, J. (1999) What software reality is really about. *IEEE Computer*, **32**, 148–149.
- Bansler, J. & Bødker, K. (1993) A reappraisal of structured analysis. Design in an organizational context. *ACM Transactions on Information Systems*, **11**, no. 2, 165–193.
- Bjerknes, G. (1989) *Contradictions – A Tool to Understand Situations in Systems Development*. PhD Thesis, Oslo University, Oslo, Norway (in Norwegian).
- Bjerknes, G. (1991) Dialectical reflection in information systems development. *Scandinavian Journal of Information Systems*, **3**, 55–77.
- Bourque, P., Dupuis, R., Abran, A., Moore, J. & Tripp, L. (1999) The guide to the software engineering body of knowledge. *IEEE Software*, **Nov/Dec**, 35–44.
- Brown, J.S., Collins, A. & Duguid, S. (1989) Situated cognition and the culture of learning. *Educational Researcher*, **18**, no. 1, 32–42.
- Brown, J.S. & Duguid, P. (1991) Organizational Learning and Communities-of-Practice. *Organization Science*, **2**, no. 1.
- Bruner, J. (1966) *Toward a Theory of Instruction*. Harvard University Press, Cambridge, MA.
- Clark, M.A.C. & Boyle, R.D. (1999) A personal theory of teaching computing through final year projects. *Computer Assisted Language Learning*, **9**, no. 3, 200–214.
- Couger, J.D., Davis, G.B., Dologite, D.G., Feinstein, D.L. et al. IS '95 (1995) Guideline for undergraduate IS curriculum. *MIS Quarterly*, **19**, 341–359.
- Curtis, B., Krasner, H. & Iscoe, N. (1988) A field study of the software design process for large systems. *Communications of the ACM*, **31**, no. 11, 1268–1287.
- Dahlbom, B. & Mathiassen, L. (1997) The future of our profession. *Communications of the ACM*, **40**, no. 6, 80–89.
- Denning, P.J. (1992) *Educating a New Engineer*, **35**, no. 11, 80–89.
- Denning, P.J. (1997) How we will learn. In: *Beyond Calculation, the Next 50 Years of Computing*. Denning, P.J. & Metcalfe, R.M. (eds), pp. 267–286. ACM Press, New York, NY.
- Denning, P.J. (2000) The future of the IT profession. In: *Ubiquity: An ACM IT Magazine and Forum*. At [http://www.acm.org/ubiquity/interviews/p\\_denning\\_1.html](http://www.acm.org/ubiquity/interviews/p_denning_1.html) (accessed: 9 June 2000).
- Walz, D.B., Elam, J.J., Krasner, H. & Curtis, B. (1987) A methodology for studying software design teams. An investigation of conflict behaviours in the requirements definition phases. In: *Empirical Studies of Programmers*, Olsen, G., Soloway, E. & Sheppard, S. (eds) Vol 2, pp. 83–99. Albx, Norwood, NJ.
- Fayad, M. (1997) Software development process: a necessary evil. *Communications of the ACM*, **40**, no. 9, 101–103.

- Guindon, R., Krasner, H. & Curtis, B. (1987) Breakdowns and processes during the early activities of software design by professionals. In: *Empirical Studies of Programmers. Second Workshop*. Ablex Publishing, Norwood, NJ, 65–82.
- Hagan, D., Tucker, S. & Ceddia, J. (1999) Industrial experience projects: a balance of process and product. *Computer Assisted Language Learning*, **9**, no. 3.
- Krasner, H., Curtis, B. & Iscoe, N. (1987) Communication breakdowns and boundary spanning activities of software design by professionals. In: *Empirical Studies of Programmers. Second Workshop*, pp. 47–64. Ablex Publishing, Norwood, NJ.
- Lanzara, G.F. (1983) The design process. Frames, metaphors, and games. In: *Systems Design, for, with, and by the Users*, Briefs, U. et al. (eds), pp 29–40. North-Holland, Amsterdam.
- Lave, J. (1988) *Cognition in Practice: Mind, Mathematics, and Culture in Everyday Life*. Cambridge University Press, Cambridge, UK.
- Lave, J. & Wenger, E. (1990) *Situated Learning: Legitimate Peripheral Participation*. Cambridge University Press, Cambridge, UK.
- Leidner, D.E. & Jarvenpaa, S.L. (1995) The use of information technology to enhance management school education: a theoretical view. *MIS Quarterly*, **19**, 265–291.
- Mathiassen, L., Munk-Madsen, A., Nielsen, P.A. & Stage, J. (1996) Method Engineering: Who's the Customer? In: *Proceedings of International Conference on Method Engineering*, Atlanta, USA.
- Mathiassen, L. (1999) Reflective systems development. *Scandinavian Journal of Information Systems*, **10**, no. 1, 67–118.
- MISRC (2000) *MIS Faculty Directory*. Current version maintained on the World Wide Web at <http://webfoot.csom.umn.edu/isworld/facdir/default.htm> (accessed 5 September 2000).
- Naur, P. (1985) Intuition. In: *Software Development*, Ehrig, H. et al. (eds). Lecture Notes in Computer Science 186, pp. 60–79. Springer Verlag, Berlin.
- Necco, C.R., Gordon, C.L. & Tsai, N.W. (1987) Systems analysis and design: current practices. *MIS Quarterly*, **11**, no. 4, 461–473.
- Norman, D.A. & Spohrer, J.C. (1996) Learner-centered education. *Communications of the ACM*, **39**, no. 4, 24–27.
- O'Laughlin, M. (1992) Rethinking science education: beyond Piagetian constructivism toward a sociocultural model of teaching and learning. *Journal of Research in Science Teaching*, **29**, no. 8, 791–820.
- Parnas, D.L. (1999) Software engineering programs are not computer science programs. *IEEE Software*, **Nov/Dec**, 19–30.
- Pask, G. (1975) *Conversation, Cognition, and Learning*. Elsevier, New York.
- Piaget, J. (1929) *The Child's Conception of the World*. Harcourt, Brace Jovanovich, New York.
- Russo, N.L. & Misis, M.M. (1999) An assessment of systems analysis and design courses. *Journal of Systems and Software*, **45**, no. 3, 65–73.
- Russo, N. & Stolterman, E. (1998) Uncovering the assumptions behind information systems methodologies: implications for research and practice. In: *Proceedings of the 6th European Conference on Information Systems*, Aix-En-Provence, France.
- Schenk, K.D., Vitalari, N.P. & Davis, S.K. (1998) Differences between novice and expert systems analysts: What do we know and what do we do? *Journal of Management Information Systems*, **15**, 9–50.
- Schön, D.A. (1983) *The Reflective Practitioner. How Professionals Think in Action*. Basic Books, New York.
- Schön, D.A. (1987) *Educating the Reflective Practitioner*. Jossey-Bass Publishers, San Francisco.
- Shrout, E. (1970) *A Study of Job Related Competencies Used by Information Systems Analysts*. PhD Thesis, Oklahoma State University.
- Skinner, B.F. (1968) *The Technology of Teaching*. Appleton-Century-Crofts, New York.
- Stolterman, E. (1991) *The Hidden Rationality of Design. A Study of Methods and Practices in Systems Development*. PhD Thesis, Umeå University (in Swedish).
- Stolterman, E. (1992) How systems designers think about design and methods. Some reflections based on an interview study. *Scandinavian Journal of Information Systems*, **4**, 137–150.
- Suchman, L. (1988) *Plans and Situated Actions: The Problem of Human/Machine Communication*. Cambridge University Press, Cambridge, UK.
- Tan, M. (1994) Establishing mutual understanding in systems design. An empirical study. *Journal of Management Information Systems*, **10**, no. 4, 159–182.
- Trauth, E.M., Farwell, D.W. & Lee, D. (1993) The IS expectation gap: industry expectations versus academic preparation. *MIS Quarterly*, **17**, no. 3, 293–307.
- Vitalari, N.P. (1985) Knowledge as a basis for expertise in systems analysis. An empirical study. *MIS Quarterly*, **9**, no. 3, 221–241.

- Vitalari, N.P. & Dickson, G.W. (1983) Problem solving for effective systems analysis. An experimental exploration. *Communications of the ACM*, **26**, no. 11, 948–956.
- Vygotsky, L.S. (1978) *Mind in Society*. Harvard University Press, Cambridge, MA.
- Waltz, D.B., Elam, J.J. & Curtis, B. (1993) Inside a software design team. Knowledge acquisition, sharing, and integration. *Communications of the ACM*, **36**, no. 10, 63–77.
- White, K.B. & Leifer, R. (1986) Information systems development success. Perspectives from project team participants. *MIS Quarterly*, **10**, no. 3, 214–223.
- Yarusso, L. (1992) Constructivism versus objectivism. *Performance and Instruction*, **31**, no. 4, 7–9.

## Biographies

**Lars Mathiassen** is professor of computer science at Aalborg University, Denmark. His research is in software

engineering and information systems, most of it based on close collaboration with industry. He has co-authored many books, including *Computers in Context* (Blackwell, 1993), *Object Oriented Analysis and Design* (Marko Publishing, 2000) and *Improving Software Organizations: From Principles to Practice* (Addison-Wesley, 2001).

**Sandeep Purao** is assistant professor of computer information systems at Georgia State University. His current research interests include reuse-based design, empirical investigation of individual design behaviours and information system design for electronic commerce. He has worked on projects dealing with object distribution, measurements for object-oriented design, document management and abstraction for system development knowledge.