Computer Science Education
2004, Vol. 14, No. 2, pp. 85–100

Situated Learning in Computer Science Education

Mordechai Ben-Ari
Department of Science Teaching, Weizmann Institute of Science, Rehovot, Israel

ABSTRACT

Sociocultural theories of learning such as Wenger and Lave's situated learning have been suggested as alternatives to cognitive theories of learning like constructivism. This article examines situated learning within the context of computer science (CS) education. Situated learning accurately describes some CS communities like open-source software development, but it is not directly applicable to other CS communities, especially those that deal with non-CS application areas. Nevertheless, situated learning can inform CS education by analyzing debates on curriculum and pedagogy within this framework. CS educators should closely examine professional CS communities of practice and design educational activities to model the actual activities of those communities.

1. RATIONALE

Computer science (CS) is taught primarily at the university level by people who have advanced degrees in CS, but little or no formal training in education. In contrast, researchers in faculties of education primarily investigate learning in elementary and secondary schools, which emphasize the elementary skills of reading, writing and arithmetic, and introductions to subjects of the humanities, social studies and science. Therefore, relatively few CS educators are familiar with educational theories and relatively few specialists in education have attempted to apply these theories in computer science education (CSE), though recently attempts have been made to bridge the gap (Booth, 2001; Powers & Powers, 1999).

Several years ago, I wrote a paper on *constructivism* (Ben-Ari, 2001), one of the most popular theories in education, in which I tried to summarize the theory for CS educators and to analyze the applicability of the theory in CSE.

Address correspondence to: Mordechai Ben-Ari, Department of Science Teaching, Weizmann Institute of Science, Rehovot, Israel. E-mail: moti.ben-ari@weizmann.ac.il

More and more, CS educators explicitly mention constructivism as the theoretical basis for pedagogical proposals and educational software (Gibbs, 2000; Gray, Boyle, & Smith, 1998; Hadjerrouit, 1999). Recently, I have become aware that many specialists in education have come to the conclusion that cognitive approaches – which investigate the mental processes of the individual learner – need to be supplanted, or at least supplemented, by social approaches, which investigate the effect on learners of the social interactions within the classroom and elsewhere. This paper is an attempt to analyze a leading social theory called *situated learning* (Lave & Wenger, 1991; Wenger, 1998) from the point of a CS educator. Since situated learning has been applied within mathematics education (Lave, 1988; Schoenfeld, 1992), it is not far-fetched to assume that it may have relevance to CS education.

The point of the paper is not to prove empirically that situated learning is good or bad for CSE (as if such a sweeping statement could be proved). Rather, the idea is to demonstrate certain conclusions for CSE that seem to follow from the theory; these conclusions naturally delineate a set of research projects on the appropriateness of situated learning as a framework for investigating CSE. The analysis is necessarily subjective, but it is based upon many years of experience in CS and CSE. The paper will have served its purpose if it encourages research in CSE that is explicitly designed to confirm or refute conjectures based upon situated learning or other social theories of learning.

Section 2 contains a short introduction to situated learning, as well as a discussion of what I believe to be significant limitations of situated learning as an all-encompassing approach to learning. Section 3 surveys CSE literature that has been informed by social theories of learning. Section 4 shows that, while situated learning closely captures activities in the open-source community, it is not directly applicable to other CS and CSE communities. In Section 5, I discuss how situated learning can contribute to the analysis of important issues in CSE like curriculum development, legitimacy and textbook design. The discussion is accompanied by suggestions for research projects that investigate aspects of situated learning in the context of CSE.

2. SITUATED LEARNING, PRO AND CON

2.1. A Primer on Situated Learning

In the theory of situated learning, the learner is considered to be a participant within a *community of practice* (*CoP*). Learning occurs by a process similar to

apprenticeship called *legitimate peripheral participation* (*LPP*): (a) the learner *participates* in a community of practice, (b) the learner's presence is *legitimate* in the eyes of the members of the community, and (c) initially, the learner's participation is *peripheral*, gradually expanding in scope until the learner achieves full-fledged membership in the community. The essence of LPP is that learning takes place within the community where the knowledge is used, as opposed to learning in conventional schools which teach knowledge that is *decontextualized* (Lave & Wenger, 1991, p. 40). For example, Lave conducted an anthropological study of young boys apprenticed to tailors in Africa; instead of studying decontextualized cutting and sewing, at a very early stage the boys worked directly on actual applications like sewing buttons.

In contrast, the assumption behind conventional schools is that the knowledge that is learned will prove applicable in the future, even though the knowledge is not presented within a *specific* context nor with the intention of training the student for a *specific* occupation. In elementary and high schools, teaching of subjects is not contextual, in the sense that all students learn the same mathematics, regardless of whether they intend to become research mathematicians, stock brokers or supermarket cashiers. At most, classes are composed of students of similar abilities, but within an ability grouping, learning is not contextual in terms of the students' future plans.

2.2. Limitations of Situated Learning

Before searching for nuggets of useful insight within situated learning, I find it necessary to critique the theory. (See Ben-Ari (2003) for more details of this analysis.) LPP is a form of apprenticeship and it is totally clear that every time you begin a new job, or join an institution such as a school, you must undergo a process of enculturation in order to learn both the technical content and the norms of behavior that govern the activities of members of the group. But LPP *as it is presented* in Lave and Wenger (1991) describes the entire educational process that a newcomer undergoes on her way to full-fledged membership in the community. It is hard to see how to accomplish this with students who aspire to membership in high-tech communities engaged in science, engineering, medicine and finance. Furthermore, apprenticeship requires that the future occupation of a student be determined at a very early age, whereas decontextualized education enables this decision to be deferred until after high-school or even after college. Finally, apprenticeship is susceptible to nepotism; most people consider general, public education to be a major achievement of modern democratic societies in that it allows reasonably egalitarian opportunities to enter preferred occupations:

> Japanese still feel that education, not inborn differences, is the key to individual development. What is required for a well-functioning and fair social system is to give all children, at least at the elementary school level, the same education, so that they have the same chances for success in the modern world, where hereditary occupations are much less important. (Benjamin, 1997, p. 25)

Another central tenet of situated learning is that education should emphasize "problem situations that closely resemble *real situations* in their richness and complexity so that the experience that students gain in the classroom will be transferable" (Schoenfeld, 1992, p. 365, emphasis mine). (See also Lave (1993, p. 85).) Clearly, what is a real situation will depend decisively on the students' background and future plans, amplifying the tendency towards premature determination of an occupation. It is also clear that it is impossible to present young students with situations that are really "real": they can hardly be expected to design a cell phone or a personal digital assistant. Finally, it would be unreasonable, as well as terribly inefficient, to burden masters with the entire education of young students, as advocated by Lave (1993, p. 86–87): "This suggests a possible long-term goal for math education: engaging children in early phases of an apprenticeship to mathematical masters, so that they learn how to do what mathematicians do, or at least, experience a way in how they do it...." Especially in a rapidly changing field like CS, the specific content of secondary and even undergraduate education can become rapidly outdated, so that many educators feel that CSE must emphasize "enduring principles" (Shaw, 2000).

3. PREVIOUS WORK

Hundhausen (2002) explicitly mentions situated learning as a guiding principle of his research on algorithm visualization. He showed that learning improved when he asked students to create *presentations* of algorithms, which are activities that are central to the practice of the CoP of CS educators. Clark and Boyle (1999) support the use of final year projects, because they are authentic activities that enable enculturation of students about to graduate. Nevertheless, their concept of enculturation is diametrically opposite that of LPP! They admit

that undergraduates rarely work together with researchers, and they view undergraduate research as "a reward for students who have completed an apprenticeship of *non-authentic activities*, that is, lectures, laboratory sessions and examinations" (Clark & Boyle, 1999, p. 208, emphasis mine).

Booth (2001) integrates situated learning with her previous work on phenomenography in CSE. She defines three CS cultures (academic, professional and informal) and tries to align them with three orientations that students have when learning to program (problem/techniques, product/people and computer/codes, respectively). Hakkarainen, Palonen, and Paavola (2002) show that neither cognitive nor social approaches alone can account for the achievements of experts; instead, they propose a *knowledge-creation view* within which both individual cognition and social interaction contribute to the creation of expertise. They also define *innovative knowledge communities* as a specialization of the concept CoPs when applied to cutting-edge science and technology.

4. OPEN-SOURCE SOFTWARE DEVELOPMENT AS LPP

4.1. The Community of Practice of Open-Source Software Development

Computer programming is one area of modern science and technology that is sometimes learned by LPP. To use fashionable terminology: the personal computer empowered ordinary people, facilitating their abilities to use the technology without formal initiation into a priesthood of experts, who are recruited from self-perpetuating aristocracies holding degrees from decontextualized schools. Nowhere is LPP more vividly demonstrated than in the phenomenon called *open-source software development* (*OSSD*) (Raymond, 2000). In OSSD, everyone is allowed to copy and modify software, with the expectation that potentially useful modifications will be made freely available. Large software systems, in particular those associated with the Linux operating system, have been created by true CoPs, albeit virtual communities whose only medium of communication is the Internet. There are no gatekeepers and no barriers to participation in these communities. Furthermore, newcomers are not "sequestered," as all "artifacts of participation" (Lave & Wenger, 1991, p. 104) are posted on web sites.

The newcomer's participation is legitimate almost by definition of the OSSD community. It is initially peripheral, as newcomers simply observe the activity achievement of modern democratic societies in that it allows reasonably egalitarian opportunities to enter preferred occupations:

> Japanese still feel that education, not inborn differences, is the key to individual development. What is required for a well-functioning and fair social system is to give all children, at least at the elementary school level, the same education, so that they have the same chances for success in the modern world, where hereditary occupations are much less important. (Benjamin, 1997, p. 25)

Another central tenet of situated learning is that education should emphasize "problem situations that closely resemble *real situations* in their richness and complexity so that the experience that students gain in the classroom will be transferable" (Schoenfeld, 1992, p. 365, emphasis mine). (See also Lave (1993, p. 85).) Clearly, what is a real situation will depend decisively on the students' background and future plans, amplifying the tendency towards premature determination of an occupation. It is also clear that it is impossible to present young students with situations that are really "real": they can hardly be expected to design a cell phone or a personal digital assistant. Finally, it would be unreasonable, as well as terribly inefficient, to burden masters with the entire education of young students, as advocated by Lave (1993, p. 86–87): "This suggests a possible long-term goal for math education: engaging children in early phases of an apprenticeship to mathematical masters, so that they learn how to do what mathematicians do, or at least, experience a way in how they do it. . . ." Especially in a rapidly changing field like CS, the specific content of secondary and even undergraduate education can become rapidly outdated, so that many educators feel that CSE must emphasize "enduring principles" (Shaw, 2000).

3. PREVIOUS WORK

Hundhausen (2002) explicitly mentions situated learning as a guiding principle of his research on algorithm visualization. He showed that learning improved when he asked students to create *presentations* of algorithms, which are activities that are central to the practice of the CoP of CS educators. Clark and Boyle (1999) support the use of final year projects, because they are authentic activities that enable enculturation of students about to graduate. Nevertheless, their concept of enculturation is diametrically opposite that of LPP! They admit

that undergraduates rarely work together with researchers, and they view undergraduate research as "a reward for students who have completed an apprenticeship of *non-authentic activities*, that is, lectures, laboratory sessions and examinations" (Clark & Boyle, 1999, p. 208, emphasis mine).

Booth (2001) integrates situated learning with her previous work on phenomenography in CSE. She defines three CS cultures (academic, professional and informal) and tries to align them with three orientations that students have when learning to program (problem/techniques, product/people and computer/codes, respectively). Hakkarainen, Palonen, and Paavola (2002) show that neither cognitive nor social approaches alone can account for the achievements of experts; instead, they propose a *knowledge-creation view* within which both individual cognition and social interaction contribute to the creation of expertise. They also define *innovative knowledge communities* as a specialization of the concept CoPs when applied to cutting-edge science and technology.

4. OPEN-SOURCE SOFTWARE DEVELOPMENT AS LPP

4.1. The Community of Practice of Open-Source Software Development

Computer programming is one area of modern science and technology that is sometimes learned by LPP. To use fashionable terminology: the personal computer empowered ordinary people, facilitating their abilities to use the technology without formal initiation into a priesthood of experts, who are recruited from self-perpetuating aristocracies holding degrees from decontextualized schools. Nowhere is LPP more vividly demonstrated than in the phenomenon called *open-source software development* (*OSSD*) (Raymond, 2000). In OSSD, everyone is allowed to copy and modify software, with the expectation that potentially useful modifications will be made freely available. Large software systems, in particular those associated with the Linux operating system, have been created by true CoPs, albeit virtual communities whose only medium of communication is the Internet. There are no gatekeepers and no barriers to participation in these communities. Furthermore, newcomers are not "sequestered," as all "artifacts of participation" (Lave & Wenger, 1991, p. 104) are posted on web sites.

A newcomer's participation is legitimate almost by definition of the OSSD community. It is initially peripheral, as newcomers simply observe the activity on a web site, and use and test new versions of the software. Eventually, the newcomer can graduate to more central modes of participation, downloading the source code of the software, programming modifications and uploading the modified source code, in the hope that the community will find the modifications interesting and accept them. Raymond (2001) describes the process of enculturation in this community: "There's a natural progression from helping test programs to helping debug them to helping modify them." There is no observable teaching of newcomers who must learn by reading books and by studying source code, though questions posed on Internet forums may be answered by old-timers, provided they are convinced that the newcomer has made a serious attempt to find the answer in the posted documentation. Raymond (2001) continues: "You'll learn a lot this way, and generate good karma with people who will help you later on."

In the terminology of Wenger (1998, Chapter 2), enculturation into this CoP is accomplished when the newcomers are *mutually engaged* by participating in online discussions, checking each others' modifications and offering their own; the engagement is in a *joint enterprise* of developing free quality software; and they use a *shared repertoire* of operating systems, languages and compilers, as well as software tools for maintaining the community such as tools for configuration management, distribution and installation.

4.2. Limitations of Open-Source Software Development

One limitation to the general applicability of OSSD is its reliance on volunteer effort; most people expect to get paid for developing computer programs and most investors expect a return for paying them. However, I regard the OSSD model as extremely limited for other, more fundamental, reasons. First, it is extremely rare that the "artifacts of participation" are freely available, and, second, it is extremely rare for the knowledge needed for full-fledged participation within a CoP to be readily accessible. OSSD has been successful precisely when such knowledge is "encoded in artifacts" (Lave & Wenger, 1991, p. 102), that is, when a mastery of the artifacts is sufficient for successful participation. This might be called *domainless programming*, because it does not require significant knowledge outside the field of CS itself.

In industrial and commercial applications, however, "artifacts of participation" are simply not available. How exactly can OSSD possibly be relevant to the development of software for a cell phone network, a flight control system or a computerized stock exchange, if you do not have the relevant electronic systems at your disposal? More importantly, in order to

develop software for such applications, you need detailed *domain knowledge*. There is simply no world-wide "pool of potential programmers" (Raymond, 2000) with sufficient knowledge of digital signal processing, aeronautics and options pricing. Advocates of situated learning would probably consider all programmers in one such field as members of a virtual CoP, however, LPP-based education cannot be carried out in such communities because of the requirements of secrecy. Boeing and Airbus are hardly likely to encourage the creation of a CoP of programmers of flight-control systems, "mutually engaged" in a "joint enterprise," though, of course, CoPs will develop within each of the organizational units responsible for this activity. Since participation in an OSSD community seems to be as close as possible to the ideal of learning CS by LPP, it follows from these serious limitations that LPP cannot be taken as a general model for CSE.

In fact, even within the highly restricted environment of a single institution, or more exactly, a single project team, LPP fails as an educational model. A team is hardly likely to employ a person who does not have significant prerequisite knowledge (in CS or the domain or both), as demonstrated by past experience or an academic degree. Companies do not expect their master engineers to take upon themselves the basic education of newcomers that can be obtained in universities or technical courses. Only when newcomers demonstrate the requisite basic knowledge, does it become effective to educate them by LPP into the *specific artifacts* and *specialized domain knowledge* that they need to become full-fledged members of the community.

Hakkarainen et al. (2002) describe "Pekka," an outstanding engineer in a high-tech company, whose educational development was highly consistent with LPP as described by Lave and Wenger. From their description it is clear that Pekka is an unusually capable and resourceful person whose educational background is the exception that proves the rule that LPP is not a viable alternative to decontextualized schooling. Pekka himself is aware of this:

> Pekka acknowledges, however, the value of schooling because[,] in his view only a few people [can persevere] enough to learn by doing or have an opportunity to do it. (Hakkarainen et al., 2002)

4.2.1. Research Project

Does OSSD experience successfully prepare a newcomer for professional employment in other fields of software? There is no question that, for example, a newcomer who became a member of the Linux community could become successfully employed in the maintenance of Linux systems in an organization; the question is rather: Is this type of experience sufficient to succeed in other types of CS work, such as designing and building software for banks or factories?

4.2.2. Research Project

To what extent do CoPs arise among those engaged in software development of proprietary or highly domain-dependent systems? Tight communities obviously exist within individual enterprises, and very loose communities obviously exist at the level of theoretical knowledge traded in journals and conferences, but I conjecture that cross-enterprise CoPs rarely arise.

5. WHAT CAN BE LEARNED FROM SITUATED LEARNING?

I believe that decontextualized schooling will continue to be the fundamental method of CSE. Though we must reject a straightforward interpretation of situated learning, I will now attempt to demonstrate that many aspects of CSE can be retrospectively interpreted within the framework of situated learning, and therefore, the theory should be explicitly used in future discussions and research projects.

5.1. Framing Debates

Some of the most vociferous debates in CSE concern the choice of programming languages to be used in teaching; an example is the debate over the programming language to be used in the Advanced Placement Examination (Abelson et al., 1995; Bruce, 1996). The debates over language can be framed in terms of CoPs: some people believe that students should learn the languages preferred by the industrial CoP as soon as possible, while others prefer to choose languages for their technical excellence as determined by the research CS CoP, and still others choose languages for their pedagogical effectiveness as determined by the educational CS CoP.

Another debate that can be informed by the concepts of situated learning is the one between those who propose the use of special pedagogical libraries and environments, versus those who believe that students should use standard libraries and professional environments as soon as possible. See Koffman and Wolz (2001) and Proulx, Raab, and Rasala (2002) for some examples of this debate. *Both sides* of the debate have often unconsciously used terminology

related to situated learning: (a) the CoP of professional software developers uses APIs directly so students should learn them as soon as possible, but (b) the CoP of professional software developers is used to working with project-specific libraries so students should learn how to do that as soon as possible.

What is common to these two topics is that they are generally discussed without explicitly framing the arguments in terms of CoPs, and that doing so would clarify and simplify the issues. For example, Bruce (1996) claims that: "Smalltalk seems to be the language of choice for many on Wall Street," and this could be used as the starting point of a debate on whether or not to use Smalltalk in a course. I suggest that educational institutions in general, and CS departments in particular, should *explicitly* set out the relation between their courses of study and communities of practice. This would focus debates on curricular and technological issues: a department whose course of study includes practical work in industry would closely follow industrial practice in the choice of hardware and software used in teaching, while a department of a research-oriented university could emphasize more speculative topics and technology, chosen for their importance within the research CS CoP.

5.1.1. Research Project

Which is more important within the CoPs of software development: the ability to work directly within standard languages, libraries and tools, or the ability to use, develop and adapt special-purpose language, libraries and tools? Does this tendency vary among various CS CoPs?

5.2. Situated Learning and Pedagogy

The concept of communities of practice can also be used to inform pedagogy. Lave's studies showed that apprentice tailors perform finishing tasks on real garments such as sewing on buttons. The intention is that they be given a sense of participation in the real work of the CoP, rather than decontextualized tasks like sewing toy clothes. Similarly, beginning CS students could also be set to do finishing tasks like testing complex programs, writing documentation and creating installation scripts, instead of writing toy programs. This could deepen their sense of meaningful participation in the community.

CSE experts should devote time to analyzing what actually happens in real CoPs, and then to create learning activities that *simulate* such tasks as well as possible within the constraints of a school. (I emphasize the word simulate, because I want to distance myself from naive LPP that insists on "real" situations and contexts.) This has certainly been attempted in courses of software engineering and other project-based educational activities (Fincher, Petre, & Clark, 2001), but I would extend it to earlier levels of learning. Generally, project work has been taken to mean capstone projects (Clark & Boyle, 1999), but an examination of professional CS CoPs might reveal that activities characteristic of full-scale projects such as analysis and design are not truly representative of the activities that CS graduates are likely to encounter, at least at the beginning of their careers. The focus must be on *activities* characteristic of CoPs, not necessarily on *projects* as such. Such activities must be integrated as far as possible into ordinary courses, which will form the bulk of the curriculum for the foreseeable future.

5.2.1. Research Project

Which activities are most important within the CoPs of software development: modification of existing programs, construction of programs from existing building-blocks or design of programs? Or, rather, what is the relative importance of these activities as a function of a person's experience and of the particular CoP.

5.3. Legitimacy

A further pedagogical lesson that can be learned from the concept of CoPs has been discussed by Ben-David Kolikant (in press). She starts from Brousseau's concept of the teaching-learning process as a *didactical game* with its own rules: learners must make sense of the *milieu* – the games that the teacher has designed (Brousseau, 1997). From this she inferred that students must grant *legitimacy* to the teacher as a representative of the CoP to which they aspire. Ben-David Kolikant showed that, in the context of learning concurrency, some, but not all, students are able to make the transition from merely "surviving" the game (by trying to please the teacher) to adopting some of the techniques of the CoP of computing professionals. One of her conclusions was that examples and exercises – including their cover-stories – must be carefully chosen so that students will be willing to grant the teacher legitimacy as a representative of the CoP to which they aspire. Cute examples that are sometimes chosen in the hope of increasing motivation can have the paradoxical effect of reducing motivation by removing the legitimacy that must be granted to the teacher.

It is interesting that even the new Computing Curricula 2001 explicitly address legitimacy: in discussing a functional-first style of teaching, they see as a disadvantage the fact that: "students may react skeptically to learning a language that they see as outside the mainstream" (ACM/IEEE-CS, 2001, p. 31). I would be interested to know if physics students object to solving problems concerning inclined planes, because they are "outside the mainstream" of what physicists do today. I conjecture that they do not, because they regard their teachers as legitimate representatives of that community, so that if teachers say that you have to gain proficiency in solving exercises concerning inclined planes, then so be it. In CS, teachers have not yet attained this level of automatic legitimacy. I am sure that if we ask physics students who their heroes are, we will hear the names of academic physicists like Albert Einstein and Richard Feynmann, while if we ask the same question of CS students, we will not hear the names of Edsger Dijkstra and Leslie Lamport, but rather of college dropouts Bill Gates and Michael Dell and of hacker extraordinaire Linus Torvalds. Until perceptions of the legitimacy of CS educators change, it will be difficult to resist efforts to turn CSE into training for fashionable skills.

5.3.1. Research Project

Compare the perceptions of the legitimacy of teachers and curricula held by physics students with those held by CS students.

5.4. The CS Curriculum

The most important lesson that I draw from analyzing situated learning in the context of CSE is the importance of domain knowledge in most of the CoPs that students are likely to join. I believe that in many CoPs that use computers intensively, CS graduates are actually at a disadvantage! I was once told by an aerospace project manager that, in his experience, it is easier to train a physics graduate in CS and programming than it is to train a CS graduate in physics. Curriculum design should be more cognizant of what Shaw calls *roles* as opposed to content: "If professional societies are to be involved, it should be to establish levels of quality and a forum for sharing curriculum examples, not govern specifics of content" (Shaw, 2000, p. 378). Situated learning supports her claim that students should choose a specialization that is oriented either to an application area or to CS technical expertise or looking to future managerial responsibility.

"In its early years, computer science had to struggle for legitimacy in many institutions" (ACM/IEEE-CS, 2001, p. 11), and I conjecture that this struggle has caused CS to move too far from its roots in applications. We have added more and more pure CS courses like artificial intelligence, networking, databases, graphics and so on. Section 9.1.3 of Computing Curricula 2001 gives lip-service to "familiarity with applications," but nevertheless, the curricula are top-heavy with pure CS courses. I would like to propose that university-level CSE be constructed to take into account the wide variety of CoPs that students are likely to encounter. One track would consist of pure CS for those students interested in pursuing academic careers or careers in computer companies. But the majority of students should be counseled to take a course of study that combines roughly equal amounts of three types of courses: (a) basic grounding in CS, (b) studies taken from an applications area from science, engineering or economics, and (c) *CS studies that are relevant to the CoPs of that area.*

The basic grounding in CS would consist of principles that are unrelated to specific applications: theory, programming, algorithms, architecture and systems (CC2001 categories DS, PF, AL, AR, OS, PL). The studies in the applications area are of course outside the scope of the CS curriculum; these studies are similar to those that would be pursued by a CS student taking a minor in another subject. It is in the CS studies relevant to a CoP that the perspective changes: I propose that advanced CS studies be clustered by CoPs. For example, a student intending on joining the CoP of business and finance would take courses in HCI and information management (categories HC and IM), while a student looking to work in engineering companies would take courses in software engineering, graphics and numerical computation (categories GV, SE, CN). Certainly such choices are possible today, but they do not appear as formal offerings based upon detailed analyses of the CS curriculum performed by experts in CS, CSE and the application areas. The difficulties of offering such coherent programs would also mean that all but the largest universities would have to specialize in CS education for a limited number of CoPs.

Furthermore, I propose that the CS studies themselves be based as much as possible on real problems from the application area, not in the naive sense presented by proponents of situated learning (Lave & Wenger, 1991; Schoenfeld, 1992), but real in the sense that problems and activities model or simulate problems and activities from the CoP. There should be few if any lowest-common-denominator examples like computing an average grade or implementing a game.

5.4.1. Research Project

Investigate the CS hiring practices of highly domain-dependent industries like aerospace and finance. Do they prefer CS graduates or domain-knowledgable graduates for entry positions in applications software development?

5.4.2. Research Project

Does grouping by prospective CoPs improve learning in CS? This project could be carried out at a university with large introductory CS classes: assign students to tutorial sessions by their expressed interest in applications areas, and give them examples and exercises that are relevant to the corresponding CoPs.

5.5. Textbooks

I can retrospectively frame a pedagogical decision of mine in terms of CoPs. Professional programmers and software engineers rarely have the luxury of learning from textbooks. They are routinely required to work from formal definitions of protocols, interfaces, languages and architectures. Yet most textbooks chew their material until it is ground into minute pieces. My textbook *Ada for Software Engineers* (Ben-Ari, 1998) was deliberately designed for the type of learning that a professional software engineer is required to do within that CoP, namely, to work with original documents. The textbook does not attempt to explain every single detail of the language; instead, the focus is on concepts, terminology and usage. There are copious quotations from the language standard, as well as references by paragraph number to that document attached to almost every paragraph. I believe that a textbook such as this one will better prepare the learner to work within the CoP of software engineering.

We can also see the influence of CoPs in Judith Bishop's introductory textbook *Java Gently*, which comes not only in a plain vanilla version (Bishop, 1997), but also in a version for engineers and scientists (Bishop & Bishop, 2000), which is consistent with my proposal that learning activities be relevant to an intended CoP. Perhaps we can look forward to versions of *Java Gently* for economists and biologists.

Similar conclusions were reached by Shirley Booth who examined the learning of programming using phenomenography; she concluded that the best results are obtained by those students using a *structural approach*, in which the emphasis is on the interpretation of the problem within its domain. She suggests that 'teachers devis[e] problems which actually force the student out of the programming domain and into some other world – of administration, or Newtonian mechanics, or text processing, or whatever . . . ' (Booth, 1992, p. 236).

5.5.1. Research Project

Are students more successful in introductory CS course when examples and exercises are taken from specific areas of interest outside of CS? Does this effect dropout rates, especially of those without a firm commitment to CS studies? Can this be done within heterogeneous classes that are not divided by interest in an application area?

6. CONCLUSION

It is important to note that I am not claiming that CoPs do not form within organizations; to the contrary, it is clear that high-tech CoPs are not much different from the low-tech CoPs described in Lave and Wenger (1991) and Wenger (1998). What I am claiming is that situated learning as presented in their work cannot be accepted at face value, because it simply ignores the enormous gap between the world of education and the world of high-tech CoPs, which demand extensive knowledge of both CS subjects and applications areas. This gap can only be bridged by classical decontextualized teaching in high schools, colleges and universities. The research projects proposed here will be able to shed light on the applicability of situated learning to CS education in these institutions.

ACKNOWLEDGEMENTS

I would like to thank Yifat Ben-David Kolikant for her extensive comments on this paper. Part of this work was done during a sabbatical at the Department of Computer Science of the University of Joensuu, Finland.

REFERENCES

Abelson, H., Bruce, K., van Dam, A., Harvey, B., Tucker, A., & Wegner, P. (1995). Letter opposing APCS change. *Communications of the ACM*, *38*(6), 116–117. (http://www.cs.williams.edu/~kim/APCS.html)

ACM/IEEE-CS. (2001). *Computing curricula 2001: Computer science volume*. (http://www.acm.org/sigcse/cc2001)

Ben-Ari, M. (1998). *Ada for software engineers*. Chichester, UK: John Wiley & Sons.

Ben-Ari, M. (2001). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, *20*(1), 45–73.

Ben-Ari, M. (2003). Situated learning in this high-technology world. In *Seventh International History, Philosophy & Science Teaching Conference*. Winnipeg, Canada. (http://stwww.weizmann.ac.il/g-cs/benari/articles/sit-hp.rtf)

Ben-David Kolikant, Y. (in press). Learning concurrency as an entry point to the community of CS practitioners. *Journal of Computers in Mathematics and Science Teaching*.

Benjamin, G.R. (1997). *Japanese lessons: A year in a Japanese school through the eyes of an American anthropologist and her children*. New York, NY: New York Unversity Press.

Bishop, J.M. (1997). *Java gently*. Harlow, UK: Addison-Wesley.

Bishop, J.M., & Bishop, N.T. (2000). *Java gently for engineers and scientists*. Harlow, UK: Addison-Wesley.

Booth, S. (1992). *Learning to program: A phenomenographic perspective*. Göteborg, Sweden: Acta Universitatis Gothoburgensis.

Booth, S. (2001). Learning to program as entering the datalogical culture: A phenomenographic exploration. In *9th European Association for Research on Learning and Instruction Conference*. Fribourg, Switzerland. (http://www.pedu.chalmers.se/shirley/EARLI2001/Booth.pdf)

Brousseau, G. (1997). *Theory of didactical situations in mathematics*. Dordrecht: Kluwer Academic Publishers.

Bruce, K. (1996). *Protest letter concerning the proposed change to the AP CS exam to C++*. (http://www.cs.williams.edu/~kim/ETSCplus.html)

Clark, M.A.C., & Boyle, R.D. (1999). A personal theory of teaching computing through final year projects. *Computer Science Education*, *9*, 200–214.

Fincher, S., Petre, M., & Clark, M. (Eds.). (2001). *Computer science project work: Principles and pragmatics*. London: Springer-Verlag.

Gibbs, D.C. (2000). The effect of a constructivist learning environment for field-dependent/independent students on achievement in introductory computer programming. *SIGCSE Bulletin*, *32*(1), 207–211.

Gray, J., Boyle, T., & Smith, C. (1998). A constructivist learning environment implemented in Java. *SIGCSE Bulletin*, *30*(3), 94–97.

Hadjerrouit, S. (1999). A constructivist approach to object-oriented design and programming. *SIGCSE Bulletin*, *31*(3), 171–174.

Hakkarainen, K., Palonen, T., & Paavola, S. (2002). Three perspectives of studying expertise. In *The Fifth International Conference of the Learning Sciences*. Seattle, Washington.

Hundhausen, C.D. (2002). Integrating algorithm visualization technology into an undergraduate algorithms course: Ethnographic studies of a social constructivist approach. *Computers and Education*, *39*(3), 237–260.

Koffman, E., & Wolz, U. (2001). A simple Java package for GUI-like interactivity. *SIGCSE Bulletin*, *33*(1), 11–15.

Lave, J. (1988). *Cognition in practice: Mind, mathematics and culture in everyday life*. Cambridge: Cambridge University Press.

Lave, J. (1993). Word problems: A microcosm of theories of learning. In P. Light & G. Butterworth (Eds.), *Context and cognition: Ways of learning and knowing* (pp. 74–92). Hillsdale, NJ: Lawrence Erlbaum Associates.

Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*. Cambridge: Cambridge University Press.

Powers, K.D., & Powers, D.T. (1999). Making sense of teaching methods in computer education. In *Proceedings of the 29th ASEE/IEEE Frontiers in Education Conference* (pp. 30–35). San Juan, Puerto Rico. (http://fie.engrng.pitt.edu/fie99/papers/1202.pdf)

Proulx, V.K., Raab, J., & Rasala, R. (2002). Objects from the beginning – with GUIs. *SIGCSE Bulletin*, *34*(3), 65–69.

Raymond, E.S. (2000). *The cathedral and the bazaar.* (http://www.catb.org/~esr/writings/cathedral-bazaar)

Raymond, E.S. (2001). *How to become a hacker.* (http://www.catb.org/~esr/faqs/hacker-howto.html)

Schoenfeld, A.H. (1992). Learning to think mathematically: Problem solving, metacognition, and sense making in mathematics. In D. Grouws (Ed.), *Handbook for research on mathematics teaching and learning* (pp. 334–370). New York, NY: MacMillan.

Shaw, M. (2000). Software engineering education: A roadmap. In A. Finkelstein (Ed.), *The future of software engineering: Proceedings of the 22nd International Conference on Software Engineering* (pp. 371–380). New York, NY: ACM Press.

Wenger, E. (1998). *Communities of practice: Learning, meaning and identity.* Cambridge: Cambridge University Press.